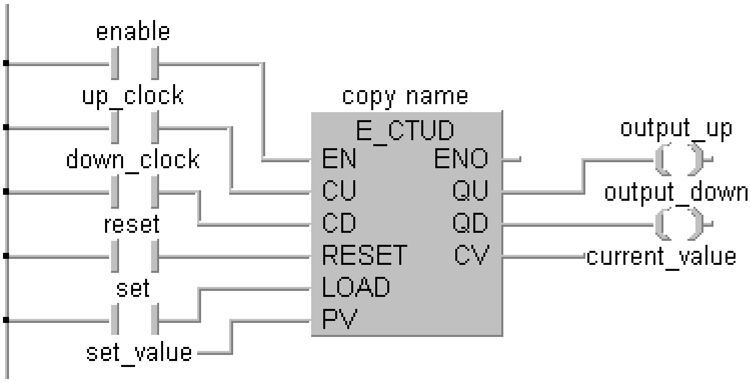
NAiS

Control FPWIN Pro

FP SERIES FP0/FP1/FP–M

Programming



Smart Solutions by NAiS

NAiS is a global brand name of Matsushita Electric Works.

NAiS FP SERIES Control FPWIN Pro FP0/FP1/FP–M Programming Matsushita Electric Works (Europe) AG ACGM0130V3.2END 5/2002

BEFORE BEGINNING

Matsushita Electric Works (Europe) AG pursues a policy of continuous improvement of the design and performance of its products, therefore, we reserve the right to change the manual/product without notice. In no event will Matsushita Electric Works (Europe) AG be liable for direct, special, incidental, or consequential damage resulting from any defect in the product or its documentation, even if advised of the possibility of such damages.

LIMITED WARRANTY

If physical defects caused by distribution are found, Matsushita Electric Works (Europe) AG will replace/repair the product free of charge. Exceptions include:

- **When physical defects are due to different usage/treatment of the product other than described in the manual.**
- **When physical defects are due to defective equipment other than the distributed product.**
- **When physical defects are due to modifications/repairs by someone other than Matsushita Electric Works (Europe) AG.**
- **When physical defects are due to natural disasters.**

Important Symbols

The following symbols are used in this manual:

⚠ **Whenever the warning triangle is used, especially important safety instructions are given. If they are not adhered to, the results could be:**

- **personal injury and/or**
- **significant damage to instruments or their contents, e.g. data**

☞ **A Note contains important additional information or indicates that you should proceed with caution.**

Example **An Example contains an illustrative example of the previous text section.**

Table of Contents

Part I

Chapter 1 Basics

Part II IEC Functions and Function Blocks

Chapter 2 Conversion Functions

Chapter 3 Numerical Functions

Chapter 4 Arithmetic Functions

Chapter 5 Process Data Type Functions

Chapter 6 Bitshift Functions

Chapter 7 Bitwise Boolean Functions

Chapter 8 Selection Functions

Chapter 9 Comparison Functions

Chapter 10 Bistable Function Blocks

Chapter 11 Edge Detection

Chapter 12 Counter

Chapter 13 Timer

Part III Matsushita Instructions

Chapter 14 Counter, Timer Function Blocks

Chapter 15 Data Transfer Instructions

Chapter 16 Arithmetic Instructions

Chapter 17 Data Comparison Instructions

Chapter 18 Logic Operation Instructions

Chapter 19 Data Shift and Rotate Instructions

Chapter 20 Data Conversion Instructions

Chapter 21 Bit Manipulation Instructions

Chapter 22 Process Control Instructions

Chapter 23 Timer Instructions

Chapter 24 Counter Instructions

Chapter 25 High–Speed Counter Special Instructions

Chapter 26 Basic Sequence Instructions

Chapter 27 Control Instructions

Chapter 28 Special Instructions

Appendix A High–Speed Counter, Pulse and PWM Output

Appendix B Special Data Registers

Appendix C Special Internal Relays

Appendix D Relays, Memory Areas and Constants

Appendix E System Registers

Appendix F Glossary

Index

Record of Changes

Part I

Chapter 1

Basics

1.1 Operands

In FPWIN Pro the following operands are available:

- in– and outputs (X/Y) as well as internal memory areas
- internal relays
- special internal relays
- timers and counters
- data registers
- special data registers
- file registers
- link registers and relays

The number of operands which are available depends on the PLC–type and its configuration. To see how many of the respective operands are available, see your hardware description.

1.1.1 Inputs/Outputs

The amount of inputs/outputs available depends on the PLC and unit type. Each input terminal corresponds to one input **X,** each output terminal corresponds to one output **Y**.

In system register 20 you set whether an output can be used once or more during the program.

☞ **Outputs which do not exist physically can be used like flags. These flags are non–holding, which means their contents will be lost, e.g. after a power failure.**

1.1.2 Internal Relays

Internal Relays are memory areas where you can store interim results. Internal relays are treated like internal outputs.

In system register no. 7 you define which internal relays are supposed to be holding/non–holding. Holding means that its values will be retained even after a power failure.

The number of available internal relays depends on the PLC type (see hardware description of your PLC).

1.1.3 Special Internal Relays

Special internal relays are memory areas which are reserved for special PLC functions. They are automatically set/reset by the PLC and are used:

- to indicate certain system states, e.g. errors

- as an impulse generator
- to initialize the system
- as ON/OFF control flag under certain conditions
  such as when some flags get a certain status if data are ready for transmission in a PLC network.

The number of special internal relays available depends on the PLC type (see hardware description of your PLC).

☞ **Special internal relays can only be read.**

1.1.4 Timers and Counters

Timers and Counters use one common memory and address area.

Define in system registers 5 and 6 how the memory area is to be divided between timers and counters and which timers/counters are supposed to be holding or non–holding. Holding means that even after a power failure all data will be saved, which is not the case in non–holding registers.

Entering a number in system register 5 means that the first counter is defined. All smaller numbers define timers.

For example, if you enter zero, you define counters only. If you enter the highest value possible, you define timers only.

In the default setting the holding area is defined by the start address of the counter area. This means all timers are holding and all counters are non–holding. You can of course customize this setting and set a higher value for the holding area, which means some of the timers, or if you prefer, all of them can be defined as holding.

In addition to the timer/counter area, there is a memory area reserved for the set value (SV) and the elapsed value (EV) of each timer/counter contact. The size of both areas is 16 bits (WORD). In the SV and EV area one INTEGER value from 0 to 32,767 can be stored.

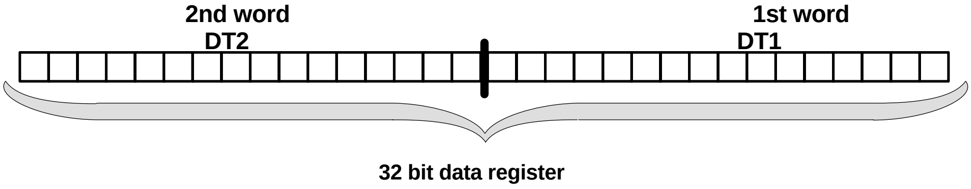
Timer/Counter No.	SV	EV	Relay
TM0	SV0	EV0	T0
. . .	. . .	. . .	. . .
TM99	SV99	EV99	T99
CT100	SV100	EV100	C100
. . .	. . .	. . .	. . .

While a timer or counter is being processed, the respective acual value can be read and under certain conditions be edited.

☞ **After changing the settings in system register 5, do not forget to adjust the addresses of the timers/counters in your PLC program because they correspond to the TM/CT numbers.**

1.1.5 Data Registers (DT)

Data registers have a width of 16 bits. You can use them, for example, to write and read constants/parameters. If an instruction requires 32 bits, two 16–bit data registers are used. If this is the case, enter the address of the first data register with the prefix DDT instead of DT. The next data register (word) will be used automatically (see example 1.2.1).



Data registers can be holding or non–holding. Holding means that even after a power failure all data will be saved. Set the holding/non–holding areas in system register 8 by entering the start address of the holding area.

The amount of data registers available depends on the PLC type (see hardware description).

1.1.6 Special Data Registers (DT)

Special data registers are like the special internal relays reserved for special functions and are in most cases set/reset by the PLC.

The register has a width of 16 bits (data type = WORD). The amount of special data registers available depends on the PLC type (see hardware description).

Most special data registers can only be read. Here some exceptions:

- actual values of the high–speed counter (DT9044 and DT9045; for FP0–T32CP DT90044 and DT90045)
- control flag of the high–speed counter DT9052 (DT90053 for FP0–T32CP)
- real–time clock (DT9054 to DT9058; FP0–T32CP: DT90054 to DT90058)
- interrupts and scan time (DT9027, DT9023–DT9024; FP0–T32CP: DT90027, DT90023–DT90024)...

1.1.7 File Registers (FL)

Some PLC types (see hardware description) provide additional data registers which can be used to increase the number of data registers. File registers are used in the same way as data registers. Set the holding/non–holding area in system register 9. Holding means that even after a power failure all data will be saved.

1.1.8 Link Relays and Registers (L/LD)

Link relays have a width of 1 bit (BOOL). In system registers 10–13 and 40–55, set the:

- transmission area
- amount of link relay words to be sent
- holding/non–holding area

Link registers have a width of 16 bits (WORD). In system registers 10–13 and 40–55, set the:

- transmission area
- amount of link relay words to be sent
- holding/non–holding area

1.2 Addresses

In the List of Global Variables, enter the physical address in the field “Address” for each global variable used in the PLC program.

The operand and the address number are part of the address. In FPWIN Pro you can use either Matsushita and/or IEC addresses. The following abbreviations are used:

Meaning	Matsushita	IEC
Input	X	I
Output	Y	Q
Memory (internal memory area)	R	M0
Timer relay	T	M1
Counter relay	C	M2
Set value	SV	M3
Elapsed value	EV	M4
Data register	DT/DDT	M5
Link relay	L	M6
Link register	LD	M7
File register	FL	M8

You find the register numbers (e.g. DT9000/DT90000) in your hardware description. The next two sections show how Matsushita and IEC addresses are composed.

1.2.1 Matsushita Addresses

A Matsushita address represents the hardware address of an in–/output, register, or counter.

For example, the hardware address of the 1st input and the 4th output of a PLC is:

- X0 (X = input, 0 = first relay)
- Y3 (Y = output, 3 = fourth relay)

Use the following Matsushita abbreviations for the memory areas. You find the register numbers in your hardware description.

Memory Area	Abbr.	Example
Memory (internal memory area)	R	R9000: self diagnostic error
Timer relay	T	T200: timer relay no. 200 (settings in system register 5+6)
Counter relay	C	C100: counter relay no. 100 (settings in system register 5+6)
Set value	SV	SV200 (set value for counter relay 200)
Elapsed value	EV	EV100 (elapsed value for timer relay 100)
Data register	DT	DT9001/DT90001 (signals power failure)
Link relay	L	L1270
Link register	LD	LD255
File register	FL	FL8188

1.2.2 IEC Addresses

The composition of an IEC–1131 address depends on:

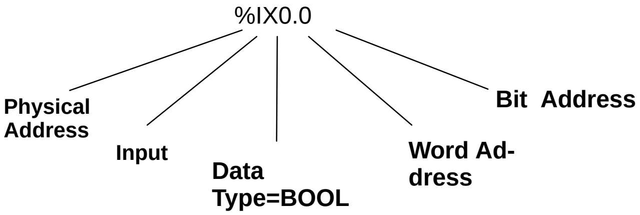
- operand type
- data type
- slot no. of the unit (word address)
- relay no. (bit address)
- PLC type

In– and Outputs are the most important components of a programmable logic controller (PLC). The PLC receives signals from the input relays and processes them in the PLC program. The results can either be stored or sent to the output relays, which means the PLC controls the outputs.

A PLC provides special memory areas, in short “M”, to store interim results, for example.

If you want to read the status of the input 1 of the first module and control the output 4 of the second module, for example, you need the physical address of each in–/output. Physical FPWIN Pro addresses are composed of the per cent sign, an abbreviation for in–/output, an abbreviation for the data type and of the word and bit address:

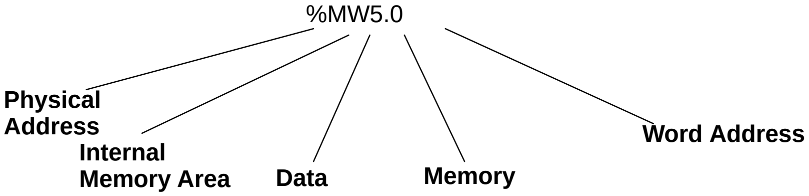
Example **IEC address for an input**



The per cent sign is the indicator of a physical address. “I” means input, “X” means data type BOOL. The first zero represents the word address (slot no.) and the second one the bit address. Note that counting starts with zero and that counting word and bit addresses differs among the PLC types.

Each PLC provides internal memory areas (M) to store interim results, for example. When using internal memory areas such as data registers, do not forget the additional number (here 5) for the memory type:

Example **IEC address for an internal memory area**



Bit addresses do not have to be defined for data registers, counters, timers, or the set and actual values.

According to IEC 1131, abbreviations for **in– and output** are “I” and “O”, respectively. Abbreviations for the **memory areas** are as follows:

Memory Type	No.	Example
Internal Relay (R)	0	%MX0.900.0 = internal relay R9000
Timer (T)	1	%MX1.200 = counter no. 200
Counter (C)	2	%MX2.100 = counter no. 100
Set Value counters/timers (SV)	3	%MW3.200 = set value of the counter no. 200
Elapsed Value counters/timers (EV)	4	%MW4.100 = elapsed value of the timer no. 100
Data Registers (DT, DDT)	5	%MW5.9001 = data register DT9001 %MD5.90001 = 32–bit data register DDT90001

☞ **Tables with hardware addresses can be found in the hardware description of your PLC.**

The following data types are available:

Data Type	Abbreviation	Range of Values	Data Width
BOOL	BOOL	0 (FALSE), 1 (TRUE)	1 bit
INTEGER	INT	–32,768 to 32,768	16 bits
DOUBLE INTEGER	DINT	–2,147,438,648 to 2,147,438,647	32 bits
WORD	WORD	0 to 65,535	16 bits
DOUBLE WORD	DWORD	0 to 4,294,987,295	32 bits
TIME 16–bit	TIME	T#0.00s to T#327.67s	16 bits*
TIME 32–bit	TIME	T#0,00s to T#21 474 836.47s	32 bits*
REAL	REAL	–1,175494 x 10^{-38} to –3,402823 x 10^{-38} and 1,175494 x 10^{-38} to 3,402823 x 10^{-38}	32 bits

*depends on your PLC

☞ **Please take into account that not all data types can be used with each IEC command.**

Numbering of in–/output addresses depends on the type of PLC used (see respective hardware description). For FP0/FP1/FP–M the addresses **are not serially numbered**. Counting restarts with zero at the first output. Supposing you have one FP1–C24 with 16 inputs and 8 outputs, the resulting addresses are: for the input: %IX0.0 – %IX0.15, and for the output: %QX0.0 – %QX0.7. In other words the counting for the word and bit number begins at zero for the outputs.

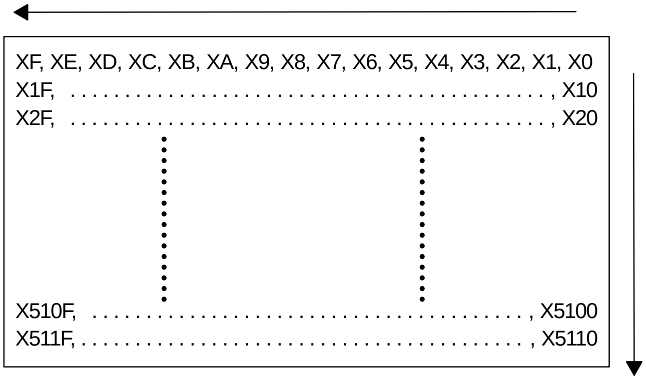
☞

- **Find the tables with all memory areas in your hardware description.**
- **When using timers, counters, set/elapsed values, and data registers, the bit address does not have to be indicated.**
- **You can also enter the register number (R9000, DT9001/90001) or the Matsushita address, e.g. “X0” (input 0), instead of the IEC address.**

1.2.3 Specifying Relay Addresses

External input relay (X), external output relay (Y), internal relay (R), link relay (L) and pulse relay (P)The lowest digit for these relay's adresses is expressed in hexadecimals and the second and higher digits are expressed in decimals as shown below.

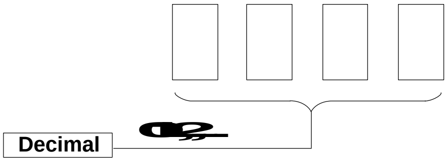
Example **Configuration of external input relay (X)**



XF, XE, XD, XC, XB, XA, X9, X8, X7, X6, X5, X4, X3, X2, X1, X0
X1F, ... , X10
X2F, ... , X20
X510F, ... , X5100
X511F, ... , X5110

1.2.4 Timer Contacts (T) and Counter Contacts (C)

Addresses of timer contacts (T) and counter contacts (C) correspond to the **TM** and **CT** instruction numbers and depend on the PLC type.



e.g. for FP2:
T0, T1 . . . . T2999
C3000, C3001 . . . . C3072

☞ **Since addresses for timer contacts (T) and counter contacts (C) correspond to the TM and CT instruction numbers, if the TM and CT instruction sharing is changed by system register 5, timer and counter contact sharing is also changed.**

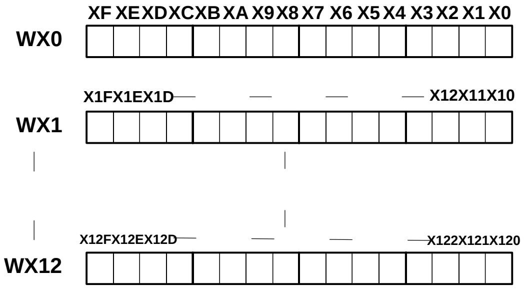
1.2.5 External Input (X) and Output Relays (Y)

- The external input relays available are those actually allocated for input use.
- The external output relays actually allocated for output can be used for turning ON or OFF external devices. The other external output relays can be used in the same way as internal relays.
- I/O allocation is based on the combination of I/O and intelligent modules installed.

1.2.6 Word Representation of Relays (WX, WY, WR, and WL)

The external input relay (X), external output relay (Y), internal relay (R) and link relay (L) can also be expressed in word format. The word format treats 16-bit relay groups as one word. The word expressions for these relays are word external input relay (WX), word external output relay (WY), word internal relay (WR) and word link relay (WL), respectively.

Example **Configuration of word external input relay (WX)**


XF XE XD XC XB XA X9 X8 X7 X6 X5 X4 X3 X2 X1 X0
WX0
X1F X1E X1D X12 X11 X10
WX1
X12F X12E X12D X122 X121 X120
WX12

☞ **Since the contents of the word relay correspond to the state of its relays (components), if some relays are turned ON, the contents of the word change.**

1.3 Constants

A constant represents a fixed value. Depending on the application, a constant can be used as a addend, multiplier, address, in–/output number, set value, etc.

There are 3 types of constants:

- decimal
- hexadecimal
- BCD

1.3.1 Decimal Constants

Decimal constants can have a width of either 16 or 32 bits.

Range 16 bit: –32,768 to 32,768

Range 32 bit: –2,147,483,648 to 2,147,483,648

Constants are internally changed into 16–bit binary numbers including character bit and are processed as such. Simply enter the decimal number in your program.

1.3.2 Hexadecimal Constants

Hexadecimal constants occupy fewer digit positions than binary data. 16 bit constants can be represented by 4–digit, 32–bit constants by 8–digit hecadecimal constants.

Range 16 bit: 8000 to 7FFF

Range 32 bit: 80000000 to 7FFFFFFFF

Enter e.g.: 16#7FFF for the hexadecimal value 7FFF in your program.

1.3.3 BCD Constants

BCD is the abbreviation for Binary Coded Decimal.

Range 16 bit: 0 to 9999

Range 32 bit: 0 to 99999999

Enter BCD constants in the program either as:

binary:	2#0001110011100101 or
hexadecimal:	16#9999

1.4 Data Types

FPWIN Pro provides elementary and user defined data types.

Elementary data types

Data Type	Abbreviation	Value Range	Data Width
BOOL	BOOL	0 (FALSE) or 1 (TRUE)	1 bit
INTEGER	INT	–32,768 to 32,768	16 bits
DOUBLE INTEGER	DINT	–2,147,483,648 to 2,147,483,647	32 bits
WORD	WORD	0 to 65,535	16 bits
DOUBLE WORD	DWORD	0 to 4,294,967,295	32 bits
TIME 16– bit	TIME	T#0,00s to T#327.67s	16 bits*
TIME 32 –bit	TIME	T#0,00s to T#21 474 836,47s	32 bits*
REAL	REAL	–1,175494 x 10^{-38} to –3,402823 x 10^{-38} and 1,175494 x 10^{-38} to 3,402823 x 10^{-38}	32 bits

*depends on your PLC

A data type has to be assigned to each variable.

User defined data types

We differentiate between **array** and **D**ata **U**nit **T**ypes (DUT). An array consists of several elementary data types which are all of the same type. A DUT consists of several elementary data types but of different data types. Each represents a new data type.

1.4.1 BOOL

Variables of the data type BOOL are binary switches. They either have the status 0 or 1 and have a width of 1 bit.

The status 0 corresponds to **FALSE** and means that the variable has the status OFF.

The status 1 corresponds to **TRUE** and means that the variable has the status ON.

The default initial value, e.g. for the variable declaration in the POU header or in the List of Global Variables = 0 (FALSE). In this case the variable has the status FALSE at the moment the PLC program starts. If it should be TRUE at the start, reset the initial value to TRUE.

1.4.2 INTEGER

Variables of the data type INTEGER are integral natural numbers (without comma) and in WORD format. The range for INTEGER values is –32,768 to 32,768 (decimal).

The default intial value, e.g. for the variable declaration in the POU header or in the List of Global Variables = 0 (FALSE). You can enter INTEGER numbers in DEC, HEX– or BIN format:

Decimal	Hexadecimal	Binary
1,234	16#4D2	2#10011010010
–1,234	16#FB2E	2#1111101100101110

1.4.3 DOUBLE INTEGER

Variables of the data type DOUBLE INTEGER are 32–bit natural numbers without commas and in DOUBLD WORD format. The range for INTEGER values is –2,147,483,648 and 2,147,483,648 decimal.

The default intial value, e.g. for the variable declaration in the POU header or in the List of Global Variables, = 0 (FALSE). You can enter DOUBLE INTEGER numbers in DEC, HEX– or BIN format:

Decimal	Hexadecimal	Binary
123,456,789	16#75BCD15	2#111010110111100110100010101
–123,456,789	16#F8A432EB	2#1111100010100100001100101110

1.4.4 STRING

The data type STRING consists of a series, i.e. string, of ASCII characters. You can store a maximum of 255 characters in one string. Each character of the string is stored in a byte.

- **The data type STRING is only available for the FP–SIGMA, FP2/2SH, FP3 and FP10SH.**
- **For the PLCs FP0, FP1 and FP–M you can only enter the data type STRING as a constant in the POU body (see F95_ASC of the Matsushita Library).**
- **For detailed information, see Online Help in FPWIN Pro.**

1.4.5 WORD

A variable of the data type WORD consists of 16 bits. The states of 16 in–/outputs can be represented by one word (WORD), for example.

The default intial value, e.g. for the variable declaration in the POU header or in the List of Global Variables, = 0 (FALSE). Enter WORD values in (DEC), HEX– or BIN format:

Decimal	Hexadecimal	Binary
1,234	16#4D2	2#10011010010
–1,234	16#FB2E	2#1111101100101110

1.4.6 DOUBLE WORD

A variable of the data type DOUBLE WORD consists of 32 bits. The states of 32 in–/outputs can be represented by one DOUBLE WORD, for example.

The default intial value, e.g. for the variable declaration in the POU header or in the List of Global Variables, = 0 (FALSE). Enter numbers in (DEC), HEX– or BIN format:

Decimal	Hexadecimal	Binary
123,456,789	16#75BCD15	2#111010110111100110100010101
–123,456,789	16#F8A432EB	2#1111100010100100001100101110

1.4.7 ARRAY

An array is a combination of variables, all of which have the same data type. This combination represents a variable itself, and therefore it has to be declared. This means that in order to make an array available for the entire project, it has to be declared in the List of Global Variables. If an array is used within a POU only, declare it in the POU header only.

Data types valid for arrays are:

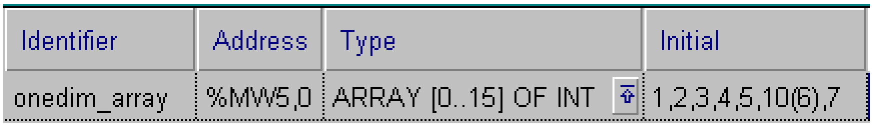
- BOOL
- INT
- DINT
- WORD
- DWORD
- TIME
- REAL

Arrays may be:

- 1–dimensional
- 2–dimensional
- 3–dimensional

Example **1–dimensional ARRAY**

Declaration in the global variable list:

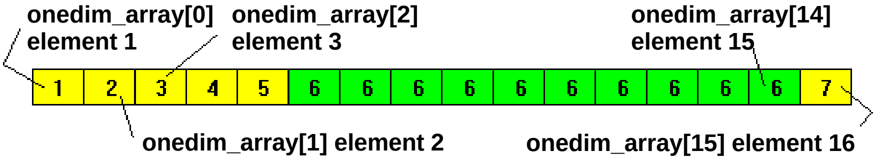


Identifier	Address	Type	Initial
onedim_array	%MW5,0	ARRAY [0..15] OF INT	1,2,3,4,5,10(6),7

Declare in the global variable list:

- identifier (name for calling up the array in the program)
- initial address where array is saved in the memory
- number of elements and data type of an array
- initial values of individual array elements and
- comment

The declared array can be imagined as follows:



1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	7

onedim_array[0] element 1
onedim_array[2] element 3
onedim_array[14] element 15
onedim_array[1] element 2
onedim_array[15] element 16

Initialize Arrays with Values

The initialisation of arrays with values starts with the first array element (*element 1*) and ends with the last array element (*element 16*). The initialisation values are entered one after another into the field *initial* and are separated from each other by commas.

If subsequent array elements are initialised with the same value, the abbreviated writing *number(value)* is possible.

* *number* stands for the number of array elements

* *value* stands for the initialisation value

In the example, *element 1* was initialised with value 1, *element 2* with value 2 etc.

Use Array Elements in the Program

You may use a 1–dimensional array element by entering *identifier[Var1]*.

* *identifier* (name of the array, see field Identifier)

* *Var1* is a variable of the type INT or a constant which has to be located in the value range of the array declaration. For this example Var1 is assigned to the range 0...15

In the example you call up the third array element (*Element 3*) with *onedim_array[2]*. If you wish to assign a value to this element in an IL program for example, you enter the following:

```
LD          current_temperature
ST          onedim_array[2]
```

Addresses of Array Elements

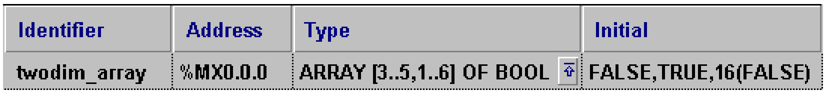
The array elements of the 1–dimensional array are subsequently saved in the PLC’s memory starting with *element 1*. This means for the example described above:

Matsushita Address	IEC–Address	Array Element	Array Element Name
DTO	%MW5.0	element 1	onedim_array(0)
DT1	%MW5.1	element 2	onedim_array(1)
DT2	%MW5.2	element 3	onedim_array(2)
DT3	%MW5.3	element 4	onedim_array(3)
DT4	%MW5.4	element 5	onedim_array(4)
...	...	...	...
DT13	%MW5.13	element 14	onedim_array(13)
DT14	%MW5.14	element 15	onedim_array(14)
DT15	%MW5.15	element 16	onedim_array(15)

Example

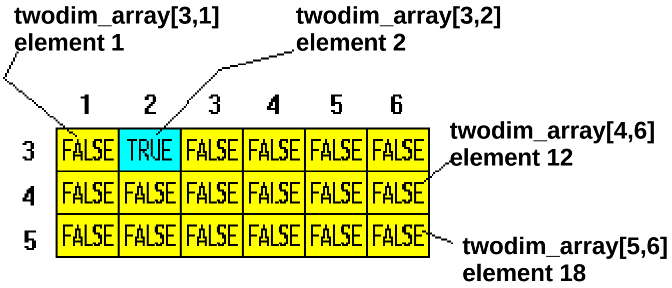
2–dimensional ARRAY

Declaration in the global variable list:



Identifier	Address	Type	Initial
twodim_array	%MX0.0.0	ARRAY [3..5,1..6] OF BOOL	FALSE,TRUE,16(FALSE)

The declared array can be imagined as follows:



	1	2	3	4	5	6
3	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
4	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
5	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Initialize arrays with values

The initialisation of arrays with values starts with the first array element (*element 1*) and ends with the last array element (*element 18*). The initialisation values are entered one after another into the field *initial* and are separated from each other by commas.

If subsequent array elements are initialised with the same value, the abbreviated writing *number(value)* is possible.

* *number* stands for the number of array elements

* *value* stands for the initialisation value

In the example *element 1* was initialised with the value FALSE, *element 2* with the value TRUE and the remaining array elements are initialised with FALSE.

Use array elements in the program

You may use a 2–dimensional array element by entering *identifier[Var1Var2]*.

* *identifier* (name of the array, see field Identifier)

* *Var1* and *Var2* are variables of the type INT or constants which have to be located in the value range of the array declaration. For this example Var1 is assigned to the range 3...5 and Var2 to the range 1...6.

In the example you call up the element 12 with *twodim_array[4,6]*. If you wish to assign a value to this element in an IL program for example, you enter the following:

```
LD          current_temperature
ST          twodim_array[4,6]
```

Addresses of array elements

The array elements of the 2–dimensional array are subsequently saved in the PLC's memory starting with *element 1*. The following storage occupation results for the example described above:

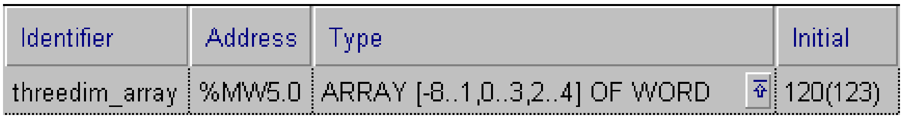
Matsushita Address	IEC–Address	Array Element	Array Element Name
R0	%MX0.0.0	element 1	twodim_array[3,1]
R1	%MX0.0.1	element 2	twodim_array[3,2]
R2	%MX0.0.2	element 3	twodim_array[3,3]
...	...	...	...
R5	%MX0.0.5	element 6	twodim_array[3,6]
R6	%MX0.0.6	element 7	twodim_array[4,1]

Matsushita Address	IEC–Address	Array Element	Array Element Name
R7	%MX0.0.7	element 8	twodim_array[4,2]
...	...	...	...
RF	%MX0.0.15	element 16	twodim_array[5,4]
R10	%MX0.1.0	element 17	twodim_array[5,5]
R11	%MX0.1.1	element 18	twodim_array[5,6]

Example

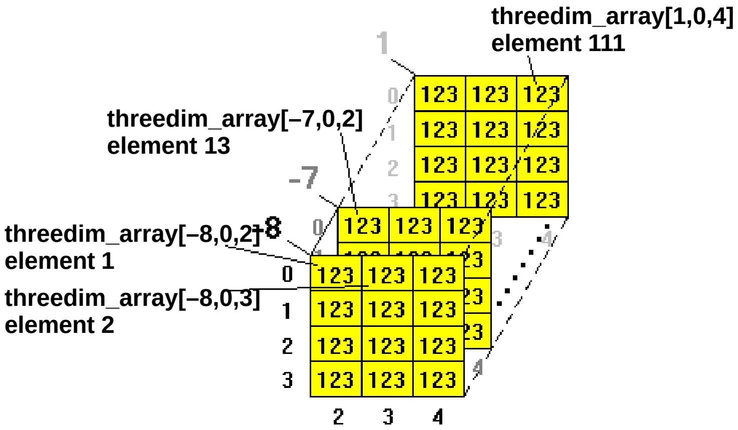
3–dimensional ARRAY

Declaration in the global variable list:



Identifier	Address	Type	Initial
threedim_array	%MW5.0	ARRAY [-8..1,0..3,2..4] OF WORD	120(123)

The declared array can be imagined as follows:



Initialize arrays with values

The initialisation of arrays with values starts with the first array element (*element 1*) and ends with the last array element (*element 120*). The initialisation values are entered one after another into the field *initial* and are separated from each other by commas. If subsequent array elements are initialised with the same value, the abbreviated writing *number(value)* is possible.

* *number* stands for the number of array elements
* *value* stands for the initialisation value

In the example all array elements were initialised with the value 123.

Use array elements in the program

Access to a 3–dimensional array is possible by entering identifier[Var1,Var2,Var3,Var4].

* *identifier* is the name of the array, (see field Identifier)

* *Var1, Var 2* and *Var3* are variables of the type INT or constants which have to be located in the value range of the array declaration (see field Type). For this example Var1 is assigned to the range 8...1 and Var2 to the range 0...3 and Var3 to the range 2...4.

In the example you call up element 15 with *threedim_array[–7,0,4]*. If you wish to assign a value to this element in an IL program, for example, you enter the following:

```
LD          current_temperature
ST          threedim_array[–7,0,4]
```

Addresses of array elements

The array elements of the 3–dimensional array are subsequently saved in the PLC's memory starting with *element 1*. The following storage occupation results for the example described above:

Matsushita Address	IEC–Address	Array Element	Array Element Name
DT0	%MW5.0	element 1	threedim_array[–8,0,2]
DT1	%MW5.1	element 2	threedim_array[–8,0,3]
DT2	%MW5.2	element 3	threedim_array[–8,0,4]
DT3	%MW5.3	element 4	threedim_array[–8,1,2]
DT4	%MW5.4	element 5	threedim_array[–8,1,3]
...	...	...	...
DT10	%MW5.10	element 11	threedim_array[–8,3,3]
DT11	%MW5.11	element 12	threedim_array[–8,3,4]
DT12	%MW5.12	element 13	threedim_array[–7,0,2]
DT13	%MW5.13	element 14	threedim_array[–7,0,3]
...	...	...	...
DT117	%MW5.117	element 118	threedim_array[1,3,2]
DT118	%MW5.118	element 119	threedim_array[1,3,3]
DT119	%MW5.119	element 120	threedim_array[1,3,4]

1.4.8 TIME

For variables of the data type TIME(32 Bit), you can indicate an interval of 0,01 to 21 474 836,47 seconds. The resolution amounts to 10ms.

Default (32–bit) = T#0   (corresponds to 0 seconds)

☞ **Times with negative signs cannot be processed. T#–2s is e.g. interpreted as T#10m53s350ms.**

Example

T#321,12s
T#321120ms
T#0,01s
T#3d5h10m3s100ms

1.4.9 REAL

Variables of the data type REAL are real numbers or floating point constants. The value range for REAL values is between –1,175494 x 10^{-38} to –3,402823 x 10^{-38} and 1,175494 x 10^{-38} to

3,402823 x 10^{-38}. The default for the initial value, e.g. for the variable declaration in the POU header or in the global variable list = 0.0 You can enter REAL values in the following format: [+–] Integer.Integer [(Ee) [+–] Integer]

Example

5.983e–7
–33.876e12
3.876e3
0.000123
123.0

☞ **The REAL value always has to be entered with a decimal point (e.g. 123.0).**

1.5 NC_TOOL Library

The NC_TOOL Library contains advanced address, information and copy functions available for all PLCs to make programming easier. Below please find a selection of these functions. For more detailed information and examples, see Online help.

Program can be adversely effected!
These functions can cause substantial problems by accessing incorrect memory areas if they are not used in the sense they were meant for. Especially other parts of the program can be adversely effected.

Name	Function
Address functions	
Adr_Of_Var_I	Address of a variable at the input of a Matsushita function
Adr_Of_Var_O	Address of a variable at the output of a Matsushita function
AdrLast_Of_Var_I	Address of a variable at the input of a Matsushita function
AdrLast_Of_Var_O	Address of a variable at the output of a Matsushita function
Adr_Of_VarOffs_I	Address of a variable with offset at the input of a Matsushita function
Adr_Of_VarOffs_O	Address of a variable with offset at the output of a Matsushita function
AdrDT_Of_Offs_I	DT address from the address offset for the input of a Matsushita function
AdrDT_Of_Offs_0	DT address from the address offset for the output of a Matsushita function
AdrFL_Of_Offs_I	FL address from the address offset for the input of a Matsushita function
AdrFL_Of_Offs_O	FL address from the address offset for the output of a Matsushita function
Functions that yield information on variables	
(E_)AreaOffs_OfVar	Yields memory area and address offset of a variable (with Enable)
(E_)Is_AreaDT	Yields TRUE if the memory area of a variable is a DT area (with Enable)
(E_)Is_AreaFL	Yields TRUE if the memory area of a variable is an FL area (with Enable)
(E_)Size_Of_Var	Yields the size of a variable in words (with Enable)
(E_)Elem_OfArray1D	Yields the number of elements in an array (with Enable)
(E_)Elem_OfArray2D	Yields the number of elements of the 1st and 2nd dimension of an array (with Enable)
(E_)Elem_OfArray3D	Yields the number of elements of the 1st, 2nd and 3rd dimension of an array (with Enable)
Additional Copy Functions	
(E_)Any16_ToBool16	Copies ANY16 to a variable with 16 elements of the data type BOOL (with Enable)
(E_)Bool16_ToAny16	Copies a variable with 16 elements of the data type BOOL to ANY16 (with Enable)
(E_)Any32_ToBool32	Copies ANY32 to a variable with 32 elements of the data type BOOL (with Enable)
(E_)Bool32_ToAny32	Copies a variable with 32 elements of the data type BOOL to ANY32 (with Enable)
(E_)Any16_ToSpecDT	Copies ANY16 to the special data register DT(9000+Offs) or DT(90000+Offs) (with Enable)
(E_)SpecDT_ToAny16	Copies the special data register DT(9000+Offs) or DT(90000+Offs) to ANY16 (with Enable)
(E_)Any32_ToSpecDT	Copies ANY32 to the special data register DT(9000+Offs) or DT(90000+Offs) (with Enable)

Name	Function
(E_)SpecDT_ToAny32	Copies the special data register DT(9000+Offs) or DT(90000+Offs) to ANY32 (with Enable)
(E_)AreaOffs_ToVar	Copies the content of an address specified by memory area and address offset to a variable (with Enable)
(E_)Var_ToAreaOffs	Copies the value of a variable to an address specified by memory area and address offset to a variable (with Enable)

Part II

IEC Functions and Function Blocks

IEC programming

For information on IEC programming and its advantages, please refer to the First Steps and IEC presentations on the installation CD for FPWIN Pro.

The difference between functions with and without enable

Functions with an enable input and output are identified by the prefix **E_**.

The ENO status (TRUE or FALSE) of the first Function (FUN) or the first function block (FB) determines whether it will be executed and whether their outputs will be written to or not .

If a subsequent FUN or FB uses one of these outputs as an input, the compiler creates a temporary variable. Since other temporary variables can occupy this address, the value is undefined at this position if it has not been written to, i.e. if ENO is FALSE.

To avoid this, make sure all FUNs or FBs in a network are executed only if the previous FUN/FB has been executed, too. The compiler simply checks that the subsequent FUN or FB has no EN input and that an AND Function is not involved.

Chapter 2

Conversion Functions

(E_)BOOL_TO_INT BOOL to INTEGER

Description

BOOL_TO_INT converts a value of the data type BOOL into a value of the data type INT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

Data types

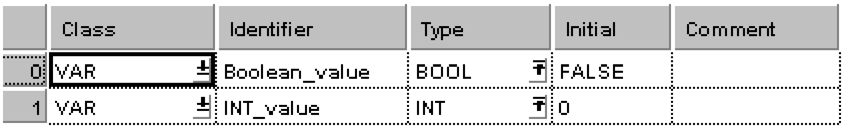
Data type	I/O	Function
BOOL	input	input data type
INT	output	converion result

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



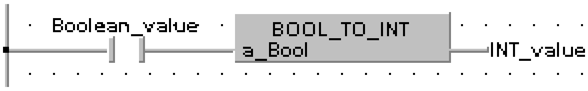
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constants for the input variables.

Body

The *Boolean_value* of the data type BOOL is converted into a value of the data type INTEGER. The converted value is written into *INT_value*.

LD



ST

```
IF Boolean_value THEN
      INT_value:=BOOL_TO_INT(Boolean_value);
END_IF;
```

(E_)BOOL_TO_DINT BOOL to DOUBLE INTEGER

Description BOOL_TO_DINT converts a value of the data type BOOL into a value of the data type DINT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

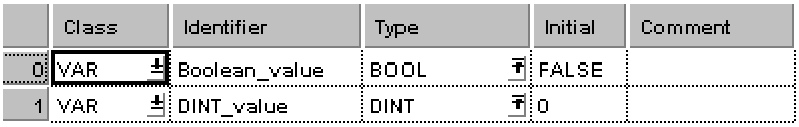
Data types

Data type	I/O	Function
BOOL	input	input data type
DINT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



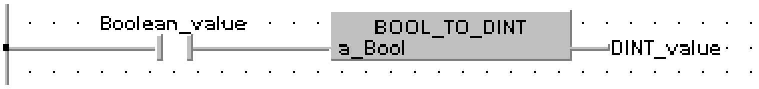
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	DINT_value	DINT	0	

This example uses variables. You may also use a constants for the input variables.

Body

The *Boolean_value* of the data type BOOL is converted into a DOUBLE INTEGER value. The converted value is written into *DINT_value*.

LD



ST

```
IF Boolean_value THEN

      DINT_value:=BOOL_TO_DINT(Boolean_value);
END_IF;
```

(E_)BOOL_TO_WORD BOOL to WORD

Description BOOL_TO_WORD converts a value of the data type BOOL into a value of the data type WORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

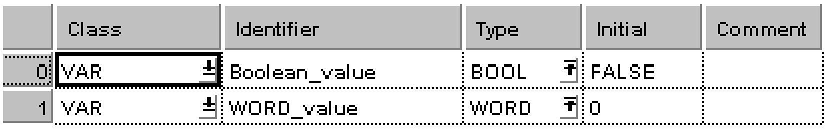
Data types

Data type	I/O	Function
BOOL	input	input data type
WORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



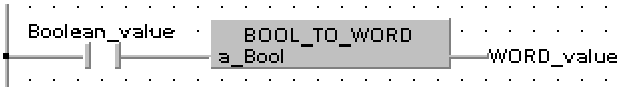
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constants for the input variables.

Body

The *Boolean_value* of the data type BOOL is converted into a value of the data type WORD. The converted value is written into *WORD_value*.

LD



ST

```
IF Boolean_value THEN

      WORD_value:=BOOL_TO_WORD(Boolean_value);
END_IF;
```

(E_)BOOL_TO_DWORD BOOL to DOUBLE WORD

Description BOOL_TO_DWORD converts a value of the data type BOOL into a value of the data type DWORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

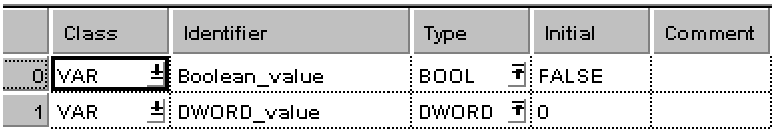
Data types

Data type	I/O	Function
BOOL	input	input data type
DWORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



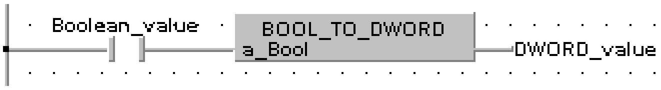
	Class	Identifier	Type	Initial	Comment
0	VAR	Boolean_value	BOOL	FALSE	
1	VAR	DWORD_value	DWORD	0	

This example uses variables. You may also use a constants for the input variables.

Body

The *Boolean_value* of the data type BOOL is converted into a value of the data type DOUBLE INTEGER. The converted value is written into *DWORD_value*.

LD



ST

```
IF Boolean_value THEN
      DWORD_value:=BOOL_TO_DWORD(Boolean_value);
END_IF;
```

(E_)BOOL_TO_STRING BOOL to STRING

Description

The function BOOL_TO_STRING converts a value of the data type BOOL to a value of the data type STRING[1]. The resulting string is represented by ’0’ or ’1’.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
BOOL	input	input data type
STRING	output	conversion result

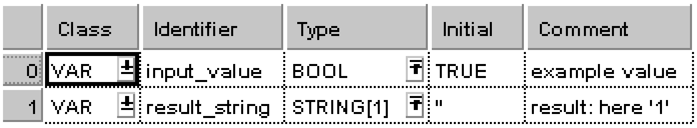
When using the data type STRING, make sure that the length of the result string is equal to or greater than the length of the source string.

Example

In this example the function BOOL_TO_STRING is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



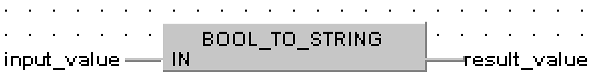
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	BOOL	TRUE	example value
1	VAR	result_string	STRING[1]	''	result: here '1'

The input variable *input_value* of the data type BOOL is intialized by the value TRUE. The output variable *result_string* is of the data type STRING[1]. It can store a maximum of one character. You can declare a character string that has more than one character, e.g. STRING[5]. From the 5 characters reserved, only 4 are used. Instead of using the variable *input_value*, you can write the constants TRUE or FALSE directly to the function’s input contact in the body.

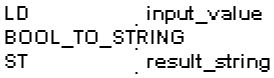
Body

The *input_value* of the data type BOOL is converted into STRING[1]. The converted value is written to *result_string*. When the variable *input_value* = TRUE, *result_string* shows ’1’.

LD



IL



```
LD              input_value
BOOL_TO_STRING
ST              result_string
```

(E_)INT_TO_BOOL INTEGER to BOOL

Description INT_TO_BOOL converts a value of the type INT into a value of the type BOOL.

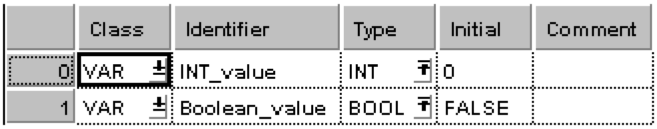
For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
INT	input	input data type
BOOL	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

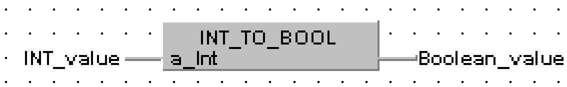


	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	Boolean_value	BOOL	FALSE	

This example uses variables. You may also use a constants for the input variables.

Body INT_value (16 bit) of the data type INTEGER is converted into a Boolean value. The result is written into *Boolean_value*.

LD



ST

```
Boolean_value:=INT_TO_BOOL(INT_value);
```

If *INT_value* has the value 0, the conversion result will be 0 (FALSE), in any other case it will be 1 (TRUE).

(E_)INT_TO_DINT INTEGER to DOUBLE INTEGER

Description INT_TO_DINT converts a value of the type INT into a value of the type DINT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

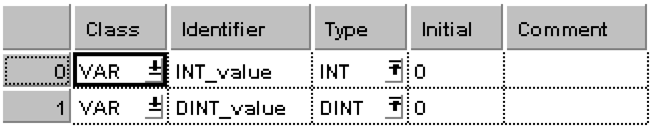
Data types

Data type	I/O	Function
INT	input	input data type
DINT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



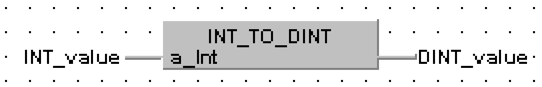
	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	DINT_value	DINT	0	

In this example the input variable (*INT_value*) has been declared. However, you may enter a constant directly at the input contact of the function.

Body

INT_value of the data type INTEGER is converted into a value of the data type DOUBLE INTEGER. The result will be written into *DINT_value*

LD



ST

```
DINT_value:=INT_TO_DINT(INT_value);
```

(E_)INT_TO_WORD INTEGER to WORD

Description INT_TO_WORD converts a value of the type INT into a value of the type WORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

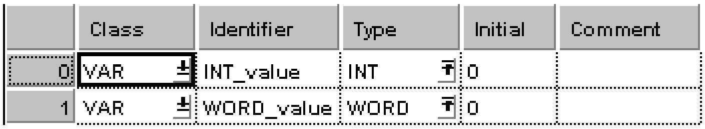
Data types

Data type	I/O	Function
INT	input	input data type
WORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



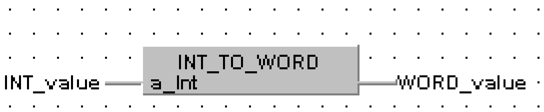
	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constants for the input variables.

Body

INT_value of the data type INTEGER is converted into a value of the data type WORD. The result is written in *WORD_value*.

LD



ST

```
WORD_value:=INT_TO_WORD(INT_value);
```

The bit combination of the input variable is assigned to the output variable.

(E_)INT_TO_DWORD INTEGER to DOUBLE WORD

Description INT_TO_DWORD converts a value of the type INT into a value of the type DWORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

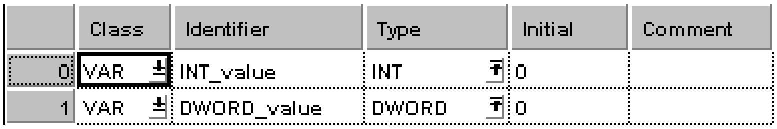
Data types

Data type	I/O	Function
INT	input	input data type
DWORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



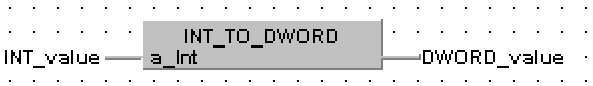
	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	DWORD_value	DWORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a value of the data type DOUBLE WORD (32 bit). The result is written in *DWORD_value*.

LD



ST

```
DWORD_value:=INT_TO_DWORD(INT_value);
```

(E_)INT_TO_REAL INTEGER to REAL

Description INT_TO_REAL converts a value of the data type INTEGER into a value of the data type REAL.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

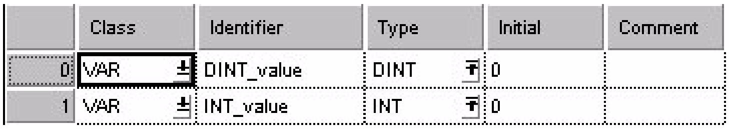
Data types

Data type	I/O	Function
INT	input	input data type
REAL	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



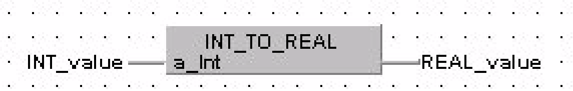
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a value of the data type REAL.The converted value is stored in *REAL_value*.

LD



ST

```
REAL_value:=INT_TO_REAL(INT_value);
```

(E_)INT_TO_TIME INTEGER to TIME

Description INT_TO_TIME converts a value of the type INT into a value of the type TIME. The resolution is 10ms, e.g. when the INTEGER value = 350, the TIME value = 3s500ms.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

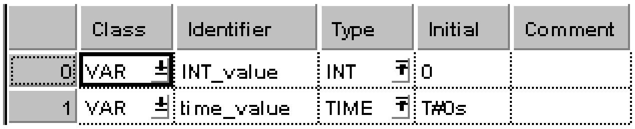
Data types

Data type	I/O	Function
INT	input	input data type
TIME	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



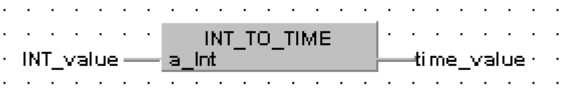
	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	time_value	TIME	T#0s	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a value of the data type TIME. The result will be written into the output variable *time_value*.

LD



ST

```
time_value:=INT_TO_TIME(INT_value);
```

(E_)INT_TO_BCD INTEGER to BCD

Description INT_TO_BCD converts a binary value of the type INTEGER in a BCD value (binary coded decimal integer) of the type WORD in order to be able to output BCD values in word format.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

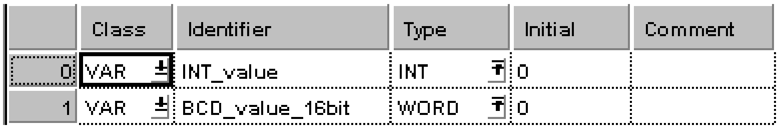
Data types

Data type	I/O	Function
INT	input	input data type
WORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



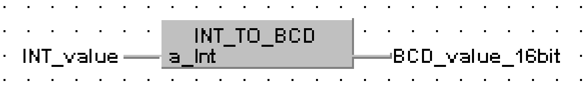
	Class	Identifier	Type	Initial	Comment
0	VAR	INT_value	INT	0	
1	VAR	BCD_value_16bit	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

INT_value of the data type INTEGER is converted into a BCD value of the data type WORD. The converted value is written into *BCD_value_16bit*.

LD



ST

```
BCD_value_16bit:=INT_TO_BCD(INT_value);
```

Since the output variable is of the type WORD and 16 bits wide, the value of the input variable should have a maximum of 4 decimal places and should thus be located between 0 and 9999.

(E_)INT_TO_STRING INTEGER to STRING

Description

The function INT_TO_STRING converts a value of the data type INT to a value of the data type STRING[6]. The resulting string is right justified within the range '–32768' to '32767'. The plus sign is omitted in the positive range. Leading zeros are filled with empty spaces (e.g. out of –12 of STRING '   –12').

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

Data types

Data type	I/O	Function
INT	input	input data type
STRING[6]	output	conversion result

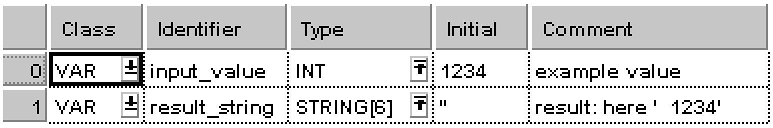
When using the data type STRING, make sure that the length of the result string is equal or greater than the length of the source string.

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



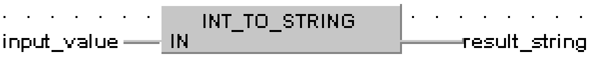
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	INT	1234	example value
1	VAR	result_string	STRING[6]	''	result: here ' 1234'

The input variable *input_value* of the data type INT is intialized by the value 1234. The output variable *result_string* is of the data type STRING[6]. It can store a maximum of 6 characters. Instead of using the variable *input_value*, you can enter a constant directly at the function's input contact in the body.

Body

The *input_value* of the data type INT is converted into STRING[6]. The converted value is written to *result_string*. When the variable *input_value* = 1234, *result_string* shows ' 1234'.

LD



ST

```
result_string:= INT_TO_STRING(input_value);
```

(E_)DINT_TO_BOOL DOUBLE INTEGER to BOOL

Description DINT_TO_BOOL converts a value of the data type DINT into a value of the data type BOOL.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

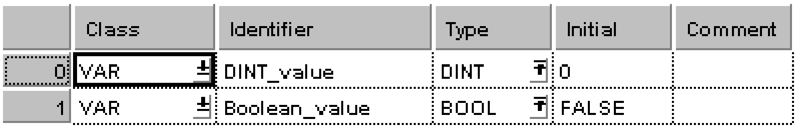
Data types

Data type	I/O	Function
DINT	input	input data type
BOOL	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



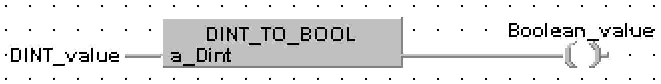
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	Boolean_value	BOOL	FALSE	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER is converted into a value of the data type BOOL. The converted value in written in *Boolean_value*.

LD



ST

```
Boolean_value:=DINT_TO_BOOL(DINT_value);
```

If the variable DINT_value has the value 0, the conversion result = FALSE, in any other case it will be TRUE.

(E_)DINT_TO_INT DOUBLE INTEGER to INTEGER

Description

DINT_TO_INT converts a value of the data type DINT into a value of the data type INT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

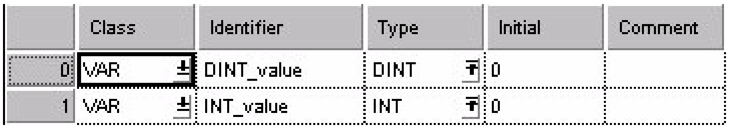
Data type	I/O	Function
DINT	input	input data type
INT	output	conversion result

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



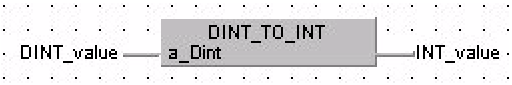
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER (32 bit) is converted into a value of the data type INTEGER (16 bit). The converted value is written in *INT_value*.

LD



ST

```
INT_value:=DINT_TO_INT(DINT_value);
```

The value of the input variable should be between –32768 and 32767.

(E_)DINT_TO_WORD DOUBLE INTEGER to WORD

Description DINT_TO_WORD converts a value of the data type DINT into a value of the data type WORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

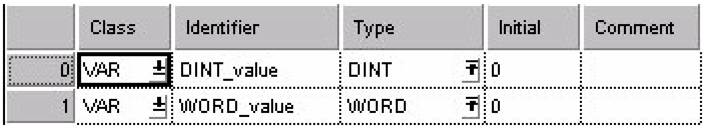
Data types

Data type	I/O	Function
DINT	input	input data type
WORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



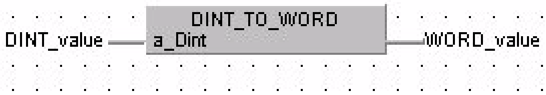
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER (32 bit) is converted into a value of the data type WORD (16 bit). The converted value is written in *WORD_value*.

LD



ST

```
WORD_value:=DINT_TO_WORD(DINT_value);
```

The first 16 bits of the input variable are assigned to the output variable.

(E_)DINT_TO_DWORD DOUBLE INTEGER to DOUBLE WORD

Description DINT_TO_DWORD converts a value of the data type DINT into a value of the data type DWORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

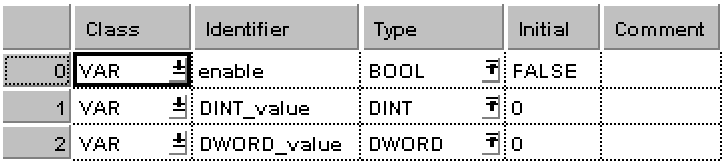
Data types

Data type	I/O	Function
DINT	input	input data type
DWORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



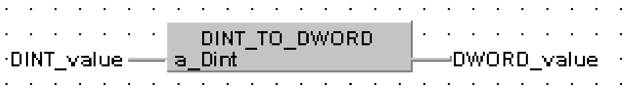
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	DINT_value	DINT	0	
2	VAR	DWORD_value	DWORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER is converted into a value of the data type DOUBLE WORD. The converted value is written in *DWORD_value*.

LD



ST

```
DWORD_value:=DINT_TO_DWORD(DINT_value);
```

The combination of the input variable is assigned to the output variable.

(E_)DINT_TO_TIME DOUBLE INTEGER to TIME

Description DINT_TO_TIME converts a value of the data type DINT into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. an input value of 123 is converted to a TIME T#1s230.00ms.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

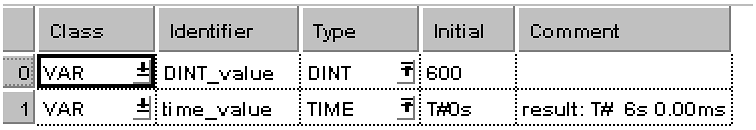
Data types

Data type	I/O	Function
DINT	input	input data type
TIME	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



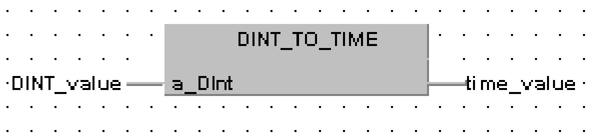
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	600	
1	VAR	time_value	TIME	T#0s	result: T# 6s 0.00ms

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER is converted to value of the data type TIME. The result is written into the output variable *time_value*.

LD



ST

```
time_value:=DINT_TO_TIME(DINT_value);
```

(E_)DINT_TO_REAL DOUBLE INTEGER to REAL

Description

DINT_TO_REAL converts a value of the data type DOUBLE INTEGER into a value of the data type REAL.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

Data types

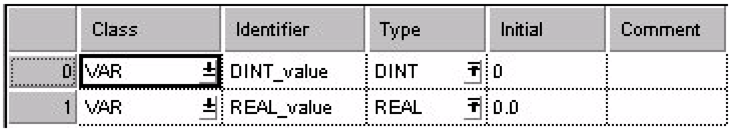
Data type	I/O	Function
DINT	input	input data type
REAL	output	conversion result

Example

In this example the function DINT_TO_REAL is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



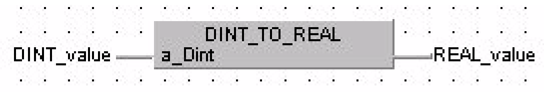
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	REAL_value	REAL	0.0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER is converted into a value of the data type REAL. The converted value is stored in *REAL_value*.

LD



ST

```
REAL_value:=DINT_TO_REAL(DINT_value);
```

(E_)DINT_TO_BCD DOUBLE INTEGER to BCD

Description DINT_TO_BCD converts a value of the data type DINT into a BCD value of the data type DWORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

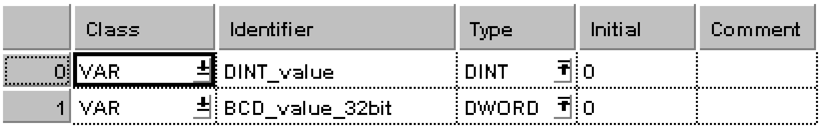
Data types

Data type	I/O	Function
DINT	input	input data type
DWORD	output	conversion result

Example In this example the function DINT_TO_BCD is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



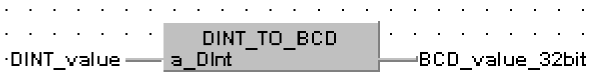
	Class	Identifier	Type	Initial	Comment
0	VAR	DINT_value	DINT	0	
1	VAR	BCD_value_32bit	DWORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

DINT_value of the data type DOUBLE INTEGER is converted into a BCD value of the data type DOUBLE WORD. The converted value is written in *BCD_value_32bit*.

LD



ST

```
BCD_value_32bit:=DINT_TO_BCD(DINT_value);
```

☞ **The value for the input variable should be between 0 and 99999999.**

(E_)DINT_TO_STRING DOUBLE INTEGER to STRING

Description The function DINT_TO_STRING converts a value of the data type DINT to a value of the data type STRING[11]. The resulting string is right justified within the range '–2147483648' to '2147483647'. The plus sign is omitted in the positive range. Leading zeros are filled with empty spaces (e.g. out of –12 of STRING '   –12').

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

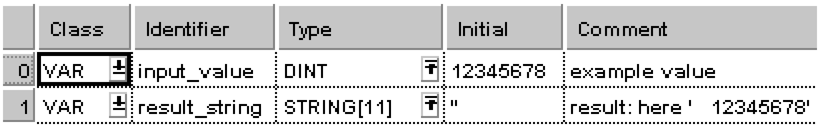
Data types

Data type	I/O	Function
DINT	input	input data type
STRING	output	conversion result

☞ **When using the data type STRING, make sure that the length of the result string is equal or greater than the length of the source string.**

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.



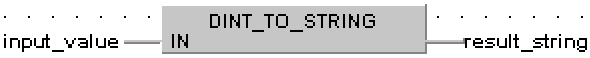
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	DINT	12345678	example value
1	VAR	result_string	STRING[11]	''	result: here ' 12345678'

The input variable *input_value* of the data type DINT is intialized by the value 12345678. The output variable *result_string* is of the data type STRING[11]. It can store a maximum of 11 characters.

Instead of using the variable *input_value*, you can enter a constant directly at the function's input contact in the body.

Body The *input_value* of the data type DINT is converted into STRING[11]. The converted value is written to *result_string*. When the variable *input_value* = 12345678, *result_string* shows '   12345678'.

LD



ST

```
result_string:=DINT_TO_STRING(input_value);
```

(E_)WORD_TO_BOOL WORD to BOOL

Description WORD_TO_BOOL converts a value of the type WORD into a value of the type BOOL.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

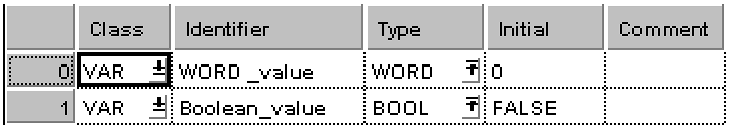
Data types

Data type	I/O	Function
WORD	input	input data type
BOOL	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



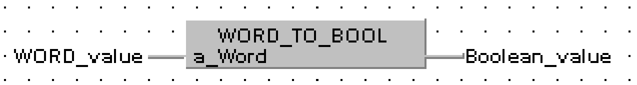
	Class	Identifier	Type	Initial	Comment
0	VAR	WORD _value	WORD	0	
1	VAR	Boolean_value	BOOL	FALSE	

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value_16bit of the data type WORD is converted into a Boolean value (~~1~~1–bit). The result will be written in *Boolean_value*.

LD



ST

```
Boolean_value:=WORD_TO_BOOL(WORD_value);
```

☞ **If the value of *WORD*_value = 0 (16#0000), the conversion result will be = 0 (FALSE), in any other case = 1 (TRUE).**

(E_)WORD_TO_INT WORD to INTEGER

Description

WORD_TO_INT converts a value of the type WORD into a value of the type INT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

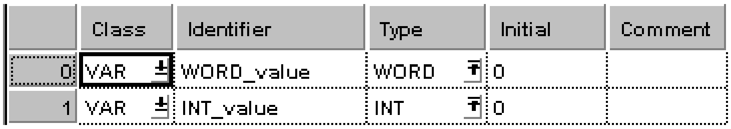
Data type	I/O	Function
WORD	input	input data type
INT	output	conversion result

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



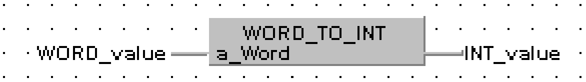
	Class	Identifier	Type	Initial	Comment
0	VAR	WORD_value	WORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constantfor the input variable.

Body

WORD_value of the data type WORD is converted into a value of the data type INTEGER. The result will be written in *INT_value*.

LD



ST

```
INT_value:=WORD_TO_INT(WORD_value);
```

The bit combination of *WORD*_value is assigned to *INT*_value.

(E_)WORD_TO_DINT WORD to DOUBLE INTEGER

Description WORD_TO_DINT converts a value of the type WORD into a value of the type DINT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

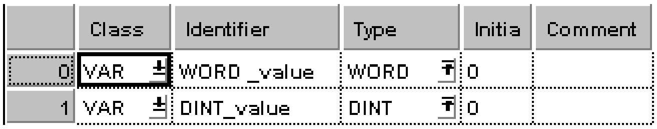
Data types

Data type	I/O	Function
WORD	input	input data type
DINT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



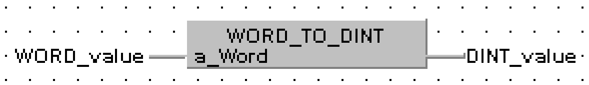
	Class	Identifier	Type	Initia	Comment
0	VAR	WORD _value	WORD	0	
1	VAR	DINT_value	DINT	0	

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD is converted into a value of the data type INTEGER. The result will be written in *DINT_value*.

LD



ST

```
DINT_value:=WORD_TO_DINT(WORD_value);
```

(E_)WORD_TO_DWORD WORD to DOUBLE WORD

Description WORD_TO_DWORD converts a value of the type WORD into a value of the type DWORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

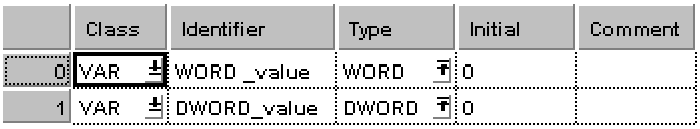
Data types

Data type	I/O	Function
WORD	input	input data type
DWORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



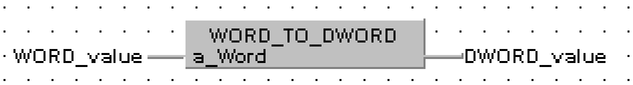
	Class	Identifier	Type	Initial	Comment
0	VAR	WORD _value	WORD	0	
1	VAR	DWORD_value	DWORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD is converted into a value of the data type DOUBLE WORD. The result will be written in *DWORD_value*.

LD



ST

```
DWORD_value:=WORD_TO_DWORD(WORD_value);
```

The bit combination of *WORD_value* is assigned to *DWORD_value*.

(E_)WORD_TO_TIME WORD to TIME

Description

WORD_TO_TIME converts a value of the type WORD into a value of the type TIME.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
WORD	input	input data type
TIME	output	conversion result

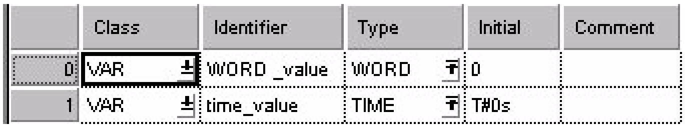
Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

input variable 12345 ⇒ output variable: T#123.45s or
input variable 16#0012 ⇒ output variable: T#180ms

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



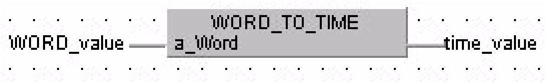
	Class	Identifier	Type	Initial	Comment
0	VAR	WORD _value	WORD	0	
1	VAR	time_value	TIME	T#0s	

This example uses variables. You may also use a constant for the input variable.

Body

WORD_value of the data type WORD (16–bit) is converted into a value of the data type TIME (16–bit). The result will be written into the output variable *time_value*.

LD



ST

```
time_value:=WORD_TO_TIME(WORD_value);
```

(E_)WORD_TO_STRING WORD to STRING

Description The function WORD_TO_STRING converts a value of the data type WORD to a value of the data type STRING[7]. In accordance with IEC–1131, the hexadecimal representation of the result string is ’16#xxxx’, whereby xxxx is the hexadecimal representation of the input value. Possible values for the result string are in the range from ’16#0000’ to ’16#FFFF’, whereby leading zeros are filled with the character zero.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

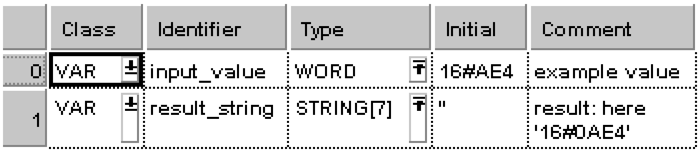
Data types

Data type	I/O	Function
WORD	input	input data type
STRING	output	conversion result

When using the data type STRING, make sure that the length of the result string is equal or greater than the length of the source string.

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

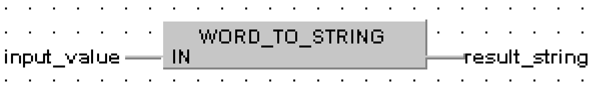


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	WORD	16#AE4	example value
1	VAR	result_string	STRING[7]	''	result: here '16#0AE4'

The input variable *input_value* of the data type WORD is intialized by the value 16#AE4. The output variable *result_string* is of the data type STRING[7]. It can store a maximum of 7 characters.
Instead of using the variable *input_value*, you can enter a constant directly at the function’s input contact in the body.

Body The *input_value* of the data type WORD is converted into STRING[7]. The converted value is written to *result_string*. When the variable *input_value* = 16#AE4, *result_string* shows ’16#0AE4’.

LD



ST

```
result_string:=WORD_TO_STRING(input_value);
```

(E_)DWORD_TO_BOOL DOUBLE WORD to BOOL

Description DWORD_TO_BOOL converts a value of the data type DOUBLE WORD into a value of the data type BOOL.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

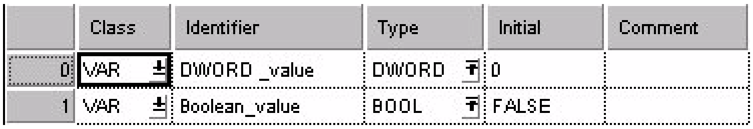
Data types

Data type	I/O	Function
DWORD	input	input data type
BOOL	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



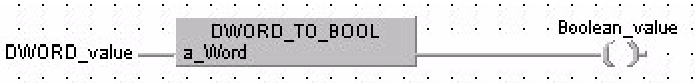
	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD _value	DWORD	0	
1	VAR	Boolean_value	BOOL	FALSE	

This example uses variables. You may also use a constant for the input variable.

Body

DWORD_value of the data type DOUBLE WORD is converted into a Boolean value (1 bit). the converted value is written in *Boolean_value.*

LD



ST

```
Boolean_value:=DWORD_TO_BOOL(DWORD_value);
```

If the variable *DWORD_value* has the value 0 (16#00000000), the conversion result will be FALSE, in any other case it will be TRUE.

(E_)DWORD_TO_INT DOUBLE WORD to INTEGER

Description DWORD_TO_INT converts a value of the data type DWORD into a value of the data type INT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

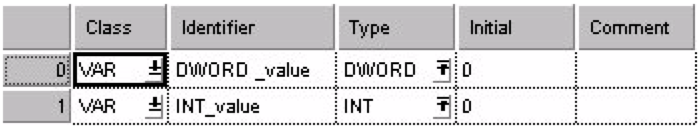
Data types

Data type	I/O	Function
DWORD	input	input data type
INT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



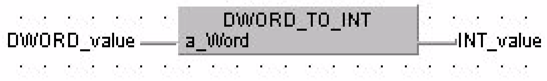
	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD _value	DWORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use constants for the input variables.

Body

DWORD_value of the data type DOUBLE WORD (32–bit) is converted into an INTEGER value (16–bit). The converted value is written in *INT_value*.

LD



ST

```
INT_value:=DWORD_TO_INT(DWORD_value);
```

The first 16 bits of the input variable are assigned to the output variable.

(E_)DWORD_TO_DINT DOUBLE WORD to DOUBLE INTEGER

Description DWORD_TO_DINT converts a value of the data type DOUBLE WORD into a value of the data type DOUBLE INTEGER.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

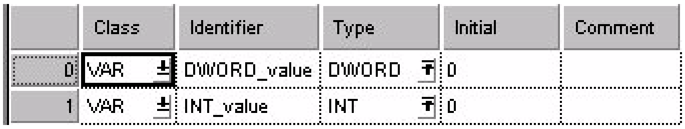
Data types

Data type	I/O	Function
DWORD	input	input data type
DINT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



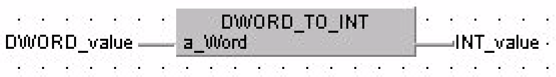
	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

DWORD_value of the data type DOUBLE WORD is converted into a DOUBLE INTEGER value. The converted value is written in *DINT_value*.

LD



ST

```
DINT_value:=DWORD_TO_DINT(DWORD_value);
```

The bit combination of the input variable will be assigned to the output variable.

(E_)DWORD_TO_WORD DOUBLE WORD to WORD

Description DWORD_TO_WORD converts a value of the data type DOUBLE WORD into a value of the data type WORD.

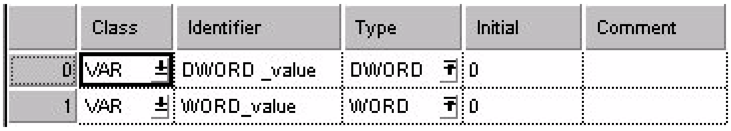
For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
DWORD	input	input data type
WORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

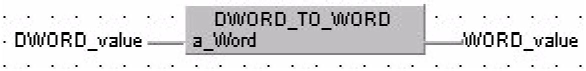


	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD _value	DWORD	0	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body *DWORD_value* of the data type DOUBLE WORD (32–bit) is converted into a value of the data type WORD (16–bit). The converted value is written in *WORD_value*.

LD



ST

```
WORD_value:=DWORD_TO_WORD(DWORD_value);
```

The first 16 bits of the input variable are assigned to the output variable.

(E_)DWORD_TO_TIME DOUBLE WORD to TIME

Description DWORD_TO_TIME converts a value of the data type DWORD into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. the input value 12345 (16#3039) is converted to a TIME T#2m3s450.00ms.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

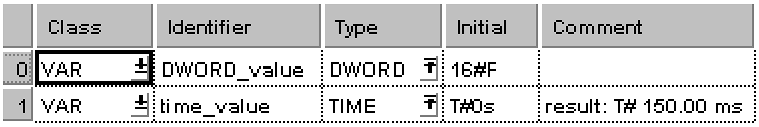
Data types

Data type	I/O	Function
DWORD	input	input data type
TIME	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



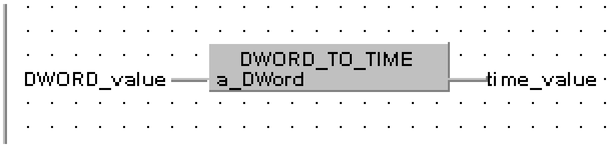
	Class	Identifier	Type	Initial	Comment
0	VAR	DWORD_value	DWORD	16#F	
1	VAR	time_value	TIME	T#0s	result: T# 150.00 ms

This example uses variables. You may also use a constant for the input variable.

Body

DWORD_value of the data type DWORD (32 bits) is converted into a value of the data type TIME (16 bits). The result is written into the output variable *time_value*.

LD



ST

```
time_value:=DWORD_TO_TIME(DWORD_value);
```

(E_)DWORD_TO_STRING DOUBLE WORD to STRING

Description

The function DWORD_TO_STRING converts a value of the data type DWORD to a value of the data type STRING[11]. In accordance with IEC–1131, the hexadecimal representation of the result string is ’16#xxxxxxxx’, whereby xxxxxxxx is the hexadecimal representation of the input value. Possible values for the result string are in the range from ’16#00000000’ to ’16#FFFFFFFF’, whereby leading zeros are filled with the character zero.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
DWORD	input	input data type
STRING	output	conversion result

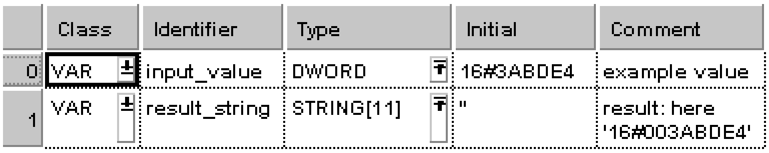
When using the data type STRING, make sure that the length of the result string is equal or greater than the length of the source string.

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



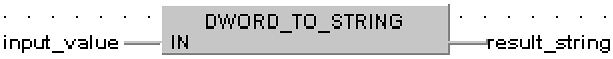
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	DWORD	16#3ABDE4	example value
1	VAR	result_string	STRING[11]	''	result: here '16#003ABDE4'

The input variable *input_value* of the data type DWORD is intialized by the value 16#3ABDE4. The output variable *result_string* is of the data type STRING[11]. It can store a maximum of 11 characters.
Instead of using the variable *input_value*, you can enter a constant directly at the function’s input contact in the body.

Body

The *input_value* of the data type DWORD is converted into STRING[11]. The converted value is written to *result_string*. When the variable *input_value* = 16#3ABDE4, *result_string* shows ’16#003ABDE4’.

LD



ST

```
result_string:=DWORD_TO_STRING(input_value);
```

(E_)REAL_TO_INT REAL to INTEGER

Description REAL_TO_INT converts a value of the data type REAL into a value of the data type INTEGER.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

This function is only available for the FP0.

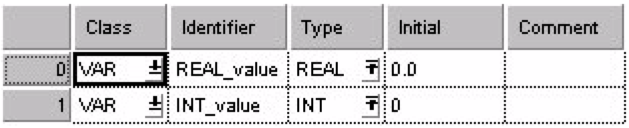
Data types

Data type	I/O	Function
REAL	input	input data type
INT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



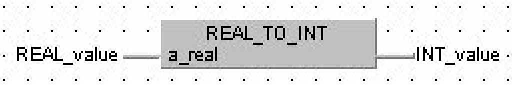
	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

REAL_value of the data type REAL is converted into a value of the data type INTEGER. The converted value is stored in *INT_value*.

LD



ST

```
INT_value:= REAL_TO_INT(REAL_value);
```

(E_)REAL_TO_DINT REAL to DOUBLE INTEGER

Description

REAL_TO_DINT converts a value of the data type REAL into a value of the data type DOUBLE INTEGER.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

Data types

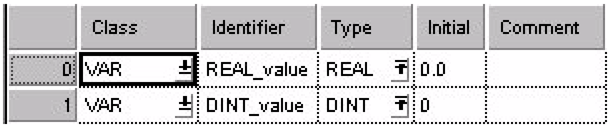
Data type	I/O	Function
REAL	input	input data type
DINT	output	conversion result

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



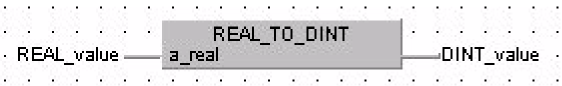
	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	
1	VAR	DINT_value	DINT	0	

This example uses variables. You may also use a constant for the input variable.

Body

REAL_value of the data type REAL is converted into a value of the data type DOUBLE INTEGER. The converted value is stored in *DINT_value*.

LD



ST

```
DINT_value:= REAL_TO_DINT(REAL_value);
```

(E_)REAL_TO_TIME REAL to TIME

Description

REAL_TO_TIME converts a value of the data type REAL to a value of the data time TIME. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when REAL = 1.0, TIME = 10ms; when REAL = 100.0, TIME = 1s. The value of the data type real is rounded off to the nearest whole number for the conversion.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

Data types

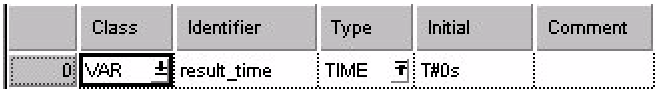
Data type	I/O	Function
REAL	input	input data type
TIME	output	conversion result

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

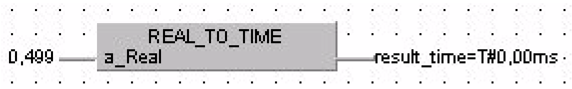


	Class	Identifier	Type	Initial	Comment
0	VAR	result_time	TIME	T#0s	

Body

By clicking on the view icon while in the online mode, you can see the result *0.00ms* immediately. Since the value at the REAL input contact is less than 0.5, it is rounded down to 0.0.

LD



ST

```
result_time:= REAL_TO_TIME(0.499);
```

(E_)REAL_TO_STRING REAL to STRING

Description The function REAL_TO_STRING converts a value from the data type REAL into a value of the data type STRING[15], which has 7 spaces both before and after the decimal point. The resulting string is right justified within the range ’–999999.0000000’ to ’9999999.0000000’. The plus sign is omitted in the positive range. Leading zeros are filled with empty spaces (e.g. out of –12.0 of STRING ’ –12.0’).

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

This function is only available for the FP0.

Data types

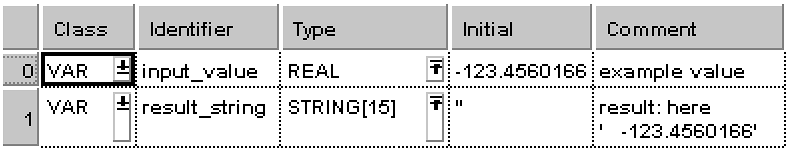
Data type	I/O	Function
REAL	input	input data type
STRING	output	conversion result

- **When using the data type STRING, make sure that the length of the result string is equal or greater than the length of the source string.**
- **The function requires approximately 160 steps of program memory. For repeated use you should integrate it into a user function that is only stored once in the memory.**

Example In this example the function REAL_TO_STRING is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



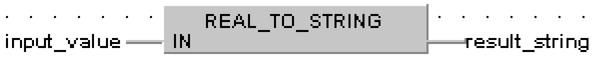
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	-123.4560166	example value
1	VAR	result_string	STRING[15]	''	result: here ' -123.4560166'

The input variable *input_value* of the data type REAL is intialized by the value –123.4560166. The output variable *result_string* is of the data type STRING[15]. It can store a maximum of 15 characters.
Instead of using the variable *input_value*, you can enter a constant directly at the function’s input contact in the body.

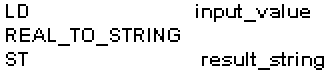
Body

The *input_value* of the data type REAL is converted into STRING[15]. The converted value is written to *result_string*. When the variable *input_value* = 123.4560166, *result_string* shows ’   –123.4560166’.

LD



IL



```
LD              input_value
REAL_TO_STRING
ST               result_string
```

(E_)TIME_TO_INT TIME to INTEGER

Description TIME_TO_INT converts a value of the type TIME into a value of the type INT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

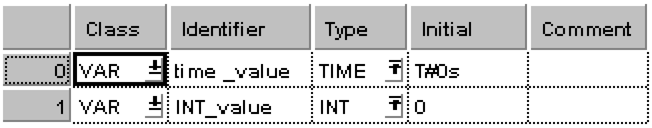
Data type	I/O	Function
TIME	input	input data type
INT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

Input variable: T#12.34s ⇒ output variable: 1234

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



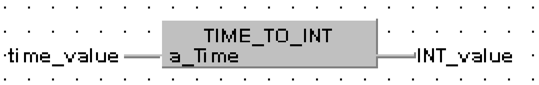
	Class	Identifier	Type	Initial	Comment
0	VAR	time _value	TIME	T#0s	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

Time_value of the data type TIME is converted into a value of the data type INTEGER. The result will be written into the output variable *INT_value*.

LD



ST

```
INT_value:=TIME_TO_INT(time_value);
```

(E_)TIME_TO_DINT TIME to DOUBLE INTEGER

Description TIME_TO_DINT converts a value of the data type TIME into a value of the data type DINT. The time 10ms corresponds to the value 1, e.g. an input value of T#1m0s is converted to the value 6000.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

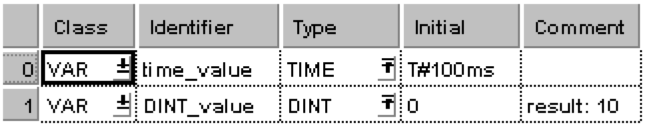
Data types

Data type	I/O	Function
TIME	input	input data type
DINT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



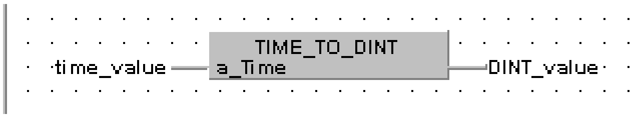
	Class	Identifier	Type	Initial	Comment
0	VAR	time_value	TIME	T#100ms	
1	VAR	DINT_value	DINT	0	result: 10

This example uses variables. You may also use a constant for the input variable.

Body

time_value of the data type TIME is converted to value of the data type DOUBLE INTEGER. The result is written into the output variable *DINT_value*.

LD



ST

```
DINT_value:=TIME_TO_DINT(time_value);
```

(E_)TIME_TO_WORD TIME to WORD

Description

TIME_TO_WORD converts a value of the type TIME into a value of the type WORD.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
TIME	input	input data type
WORD	output	conversion result

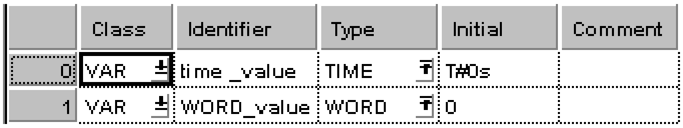
Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

Input variable: T#12.34s ⇒ output variable: 1234 or
input variable: T#1.00s ⇒ output variable: 16#0064

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

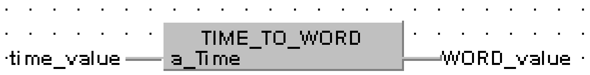


	Class	Identifier	Type	Initial	Comment
0	VAR	time_value	TIME	T#0s	
1	VAR	WORD_value	WORD	0	

This example uses variables. You may also use a constant for the input variable.

LD

Time_value of the data type TIME is converted into a value of the data type WORD. The result will be written into the output variable *WORD_value*.



ST

```
WORD_value:=TIME_TO_WORD(time_value);
```

(E_)TIME_TO_DWORD TIME to DOUBLE WORD

Description TIME_TO_DWORD bzw. E_TIME_TO_DWORD converts a value of the data type TIME into a value of the data type DWORD. The time 10ms corresponds to the value 1, e.g. an input value of T#1s is converted to the value 100 (16#64).

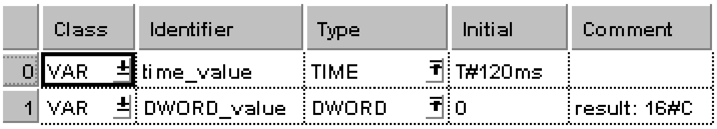
For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
TIME	input	input data type
DWORD	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

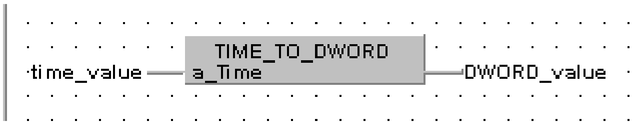


	Class	Identifier	Type	Initial	Comment
0	VAR	time_value	TIME	T#120ms	
1	VAR	DWORD_value	DWORD	0	result: 16#C

This example uses variables. You may also use a constant for the input variable.

Body *time_value* of the data type TIME is converted to a value of the data type DWORD and written into the output variable *DWORD_value*.

LD



ST

```
DWORD_value:=TIME_TO_DWORD(time_value);
```

(E_)TIME_TO_REAL TIME to REAL

Description TIME_TO_REAL converts a value of the data type TIME to a value of the data time REAL. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when TIME = 10ms, REAL = 1.0; when TIME = 1s, REAL = 100.0. The resolution amounts to 10ms.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

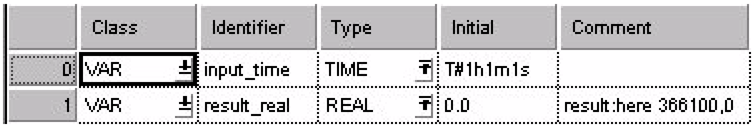
Data types

Data type	I/O	Function
TIME	input	input data type
REAL	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

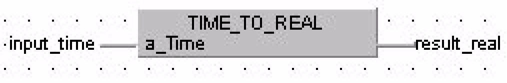
In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	input_time	TIME	T#1h1m1s	
1	VAR	result_real	REAL	0.0	result:here 366100,0

This example uses variables. You may also use a constant for the input variable.

LD



ST

```
result_real:=TIME_TO_REAL(input_time);
```

(E_)TIME_TO_STRING TIME to STRING

Description The function TIME_TO_STRING converts a value of the data type TIME to a value of the data type STRING[20]. In accordance with IEC–1131, the result string is displayed with a short time prefix and without underlines. Possible values for the result string's range are from 'T#000d00h00m00s000ms' to 'T#248d13h13m56s470ms'.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

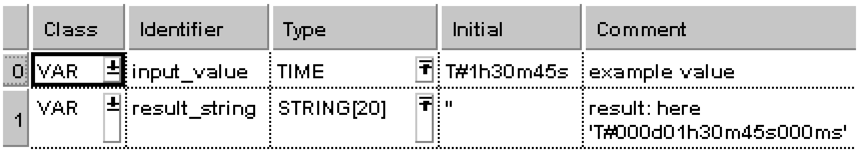
When using the data type STRING, make sure that the length of the result string is equal to or greater than the length of the source string.

Data types

Data type	I/O	Function
TIME	input	input data type
STRING	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

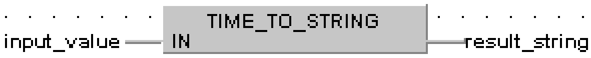


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	TIME	T#1h30m45s	example value
1	VAR	result_string	STRING[20]	''	result: here 'T#000d01h30m45s000ms'

The input variable *input_value* of the data type TIME is intialized by the value T#1h30m45s. The output variable *result_string* is of the data type STRING[20]. It can store a maximum of 20 characters.
Instead of using the variable *input_value*, you can enter a constant directly at the function's input contact in the body.

Body The *input_value* of the data type TIME is converted into STRING[20]. The converted value is written to *result_string*. When the variable *input_value* = T#1h30m45s, *result_string* shows 'T#000d01h30m45s000ms'.

LD



ST

```
result_string:=TIME_TO_STRING(input_value);
```

(E_)TRUNC_TO_INT

Truncate (cut off) decimal digits of REAL input variable, convert to INTEGER

Description

TRUNC_TO_INT cuts off the decimal digits of a REAL number and delivers an output variable of the data type INTEGER.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

- **The first 16 bits of the input variable are assigned to the output variable.**
- **Cutting off the decimal digits decreases a positive number towards zero and increases a negative number towards zero.**
- **This function is only available for the FP0.**

Data types

Data type	I/O	Function
REAL	input	input data type
INT	output	conversion result

Error flags

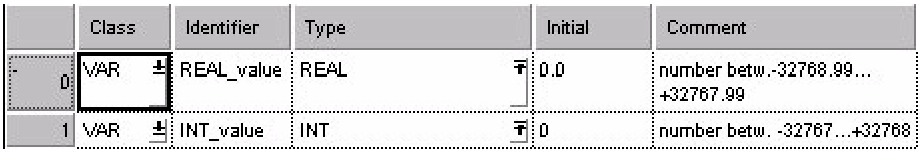
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL
R9008	%MX0.900.8	for an instant	– output variable is greater than a 16–bit INTEGER
R9009	%MX0.900.9	for an instant	– output variable is zero

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



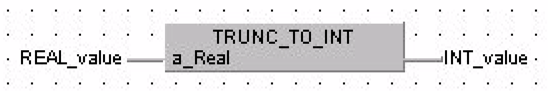
	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	number betw.-32768.99... +32767.99
1	VAR	INT_value	INT	0	number betw. -32767...+32768

This example uses variables. You may also use a constant for the input variable.

Body

The decimal digits of *REAL_value* are cut off. The result is stored as a 16–bit INTEGER in *INT_value*.

LD



ST

```
INT_value:=TRUNC_TO_INT(REAL_value);
```

(E_)TRUNC_TO_DINT

Truncate (cut off) decimal digits of REAL input variable, convert to DOUBLE INTEGER

Description TRUNC_TO_DINT cuts off the decimal digits of a REAL number and delivers an output variable of the data type DOUBLE INTEGER.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

- **This function is only available for the FP0.**
- **Cutting of the decimal digits decreases a positive number towards zero and increases a negative number towards zero.**

Data types

Data type	I/O	Function
REAL	input	input data type
DINT	output	conversion result

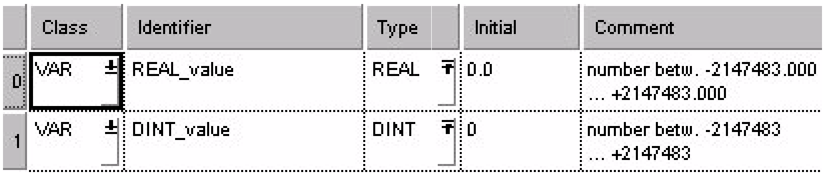
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL
R9008	%MX0.900.8	for an instant	– output variable is greater than a 32–bit DINT
R900B	%MX0.900.B	for an instant	– output variable is zero

Example In this example the function is programmed in ladder diagram (LD). You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



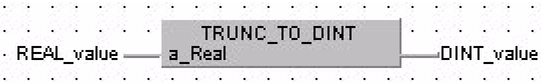
	Class	Identifier	Type	Initial	Comment
0	VAR	REAL_value	REAL	0.0	number betw. -2147483.000 ... +2147483.000
1	VAR	DINT_value	DINT	0	number betw. -2147483 ... +2147483

This example uses variables. You may also use a constant for the input variable.

Body

The decimal digits of *REAL_value* are cut off. The result is stored as a 32–bit DOUBLE INTEGER in *DINT_value*.

LD



(E_)BCD_TO_INT BCD to INTEGER

Description BCD_TO_INT converts binary coded decimal numbers (BCD) into binary values of the type INTEGER.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

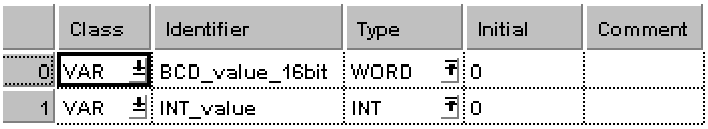
Data types

Data type	I/O	Function
WORD	input	input data type
INT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	BCD_value_16bit	WORD	0	
1	VAR	INT_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

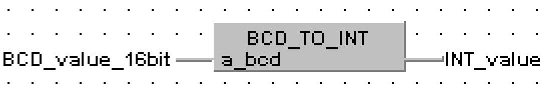
BCD constants can be indicated in FPWIN Pro as follows:

2#0001100110010101 or *16#1995*

Body

BCD_value_16bit of the data type WORD is converted into an INTEGER value. The converted value is written into output variable *INT_value*.

LD



ST

```
INT_value:=BCD_TO_INT(BCD_value_16bit);
```

(E_)BCD_TO_DINT BCD to DOUBLE INTEGER

Description BCD_TO_DINT converts a BCD value (binary coded decimal integer) of the data type DOUBLE WORD in a binary value of the data type DOUBLE INTEGER in order to process a BCD value in double word format.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

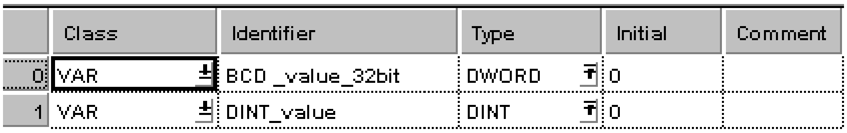
Data types

Data type	I/O	Function
DWORD	input	input data type
DINT	output	conversion result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	BCD _value_32bit	DWORD	0	
1	VAR	DINT_value	DINT	0	

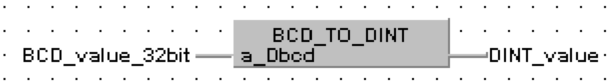
This example uses variables. You may also use a constant for the input variable.

BCD constants can be indicated in FPWIN Pro as follows:
2#00011001100101010001100110010101 or *16#19951995*

Body

BCD_value_32bit of the data type DOUBLE WORD is converted into a DOUBLE INTEGER value. The converted value is written into *DINT_value*.

LD



ST

```
DINT_value:=BCD_TO_DINT(BCD_value_32bit);
```

Chapter 3

Numerical Functions

(E_)ABS Absolute value

Description ABS calculates the value in the accumulator into an absolute value. The result is saved in the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

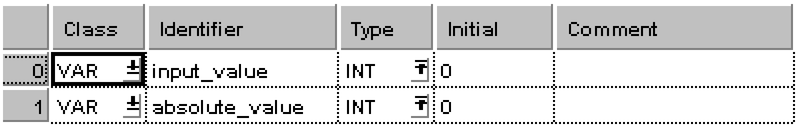
Data types

Data type	I/O	Function
INT, DINT, REAL	input	input data type
INT, DINT, REAL	output as input	absolute value

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



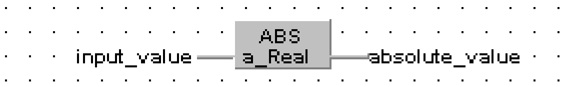
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	INT	0	
1	VAR	absolute_value	INT	0	

This example uses variables. You may also use a constant for the input variable.

Body

input_value of the data type INTEGER is converted into an absolute value of the data type INTEGER. The converted value is written in *absolute_value*.

LD



ST

```
absolute_value:=ABS(input_value);
```

Chapter 4

Arithmetic Functions

(E_)MOVE

Move value to specified destination

Description

MOVE assigns the unchanged value of the input variable to the output.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
all data types	input	source
all data types	output as input	destination

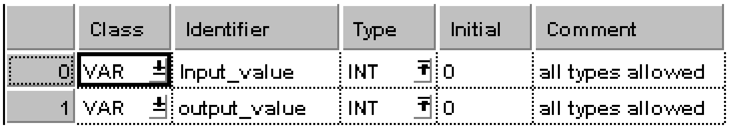
☞ **When using the data type STRING, make sure that the length of the result string is equal to or greater than the length of the source string.**

Example

In this example the function MOVE is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



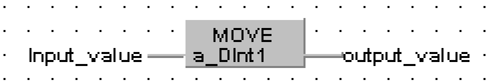
	Class	Identifier	Type	Initial	Comment
0	VAR	Input_value	INT	0	all types allowed
1	VAR	output_value	INT	0	all types allowed

This example uses variables. You may also use constants for the input variables.

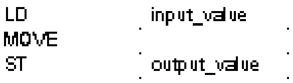
Body

Input_value is assigned to *output_value* without being modified.

LD



IL



```
LD      input_value
MOVE
ST      output_value
```

E_ADD Add

Description E_ADD adds the input variables IN1 + IN2 + ... and writes the addition result into the output variable. E_ADD operates just like the standard operator ADD (see Online Help).

Data types

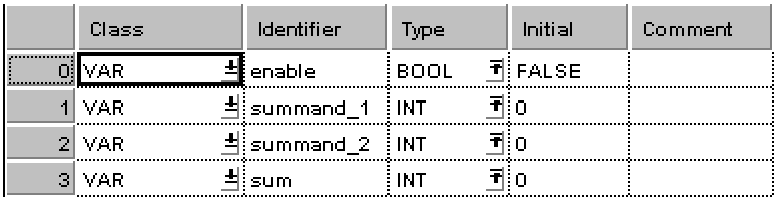
Data type	I/O	Function
INT, DINT, REAL	1st input	augend
INT, DINT, REAL	2nd input	addend
INT, DINT, REAL	output as input	sum

- **The number of input contacts a_NumN lies in the range of 2 to 28.**
- **Only the FP0 can process the data type REAL.**

Example In this example the function E_ADD is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



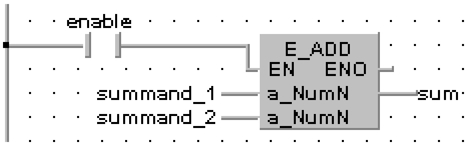
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	summand_1	INT	0	
2	VAR	summand_2	INT	0	
3	VAR	sum	INT	0	

This example uses variables. You may also use constants for the input variables.

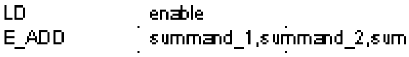
Body

If *enable* is set (TRUE), *summand_1* is added to *summand_2*. The result is written in *sum*.

LD



IL



```
LD        enable
E_ADD     summand_1,summand_2,sum
```

E_SUB Subtract

Description E_SUB operates just as the standard operator SUB (see Online Help).

Data types

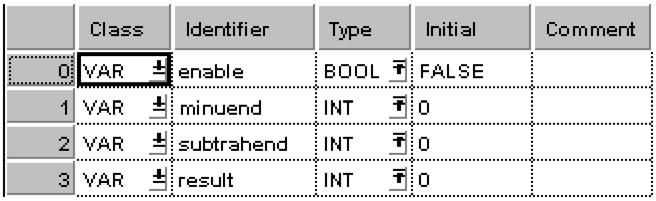
Data type	I/O	Function
INT, DINT, REAL	1st input	minuend
INT, DINT, REAL	2nd input	subtrahend
INT, DINT, REAL	output as input	result

☞ **Only the FP0 can process the data type REAL.**

Example In this example the function E_SUB is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



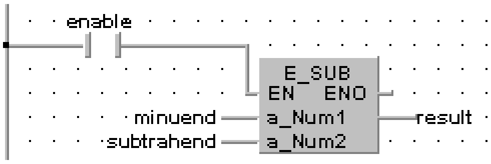
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	minuend	INT	0	
2	VAR	subtrahend	INT	0	
3	VAR	result	INT	0	

This example uses variables. You may also use constants for the input variables

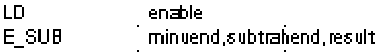
Body

If *enable* is set, *subtrahend* (data type INT) is subracted from *minuend*. The result will be written in *result* (data type INT).

LD



IL



```
LD       enable
E_SUB    minuend,subtrahend,result
```

E_MUL Multiply

Description E_MUL multiplies the values of the input variables with each other and writes the result into the output variable. E_MUL operates just as the standard operator MUL (see Online Help).

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	multiplicand
INT, DINT, REAL	2nd input	multiplicator
INT, DINT, REAL	output as input	result

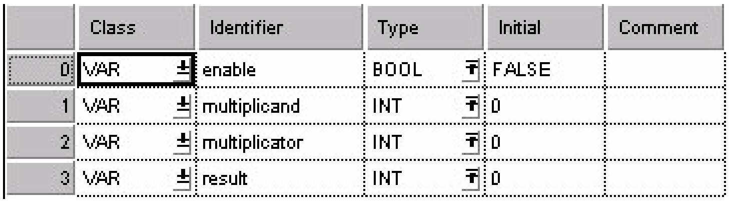
The input variables have to be of the same data type.

- **The number of input contacts a_NumN lies in the range of 2 to 28.**
- **Only the FP0 can process the data type REAL.**

Example In this example the function E_MUL is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



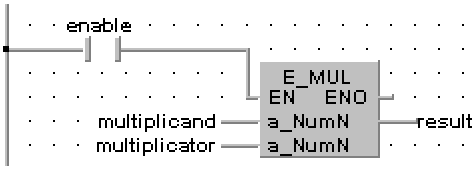
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	multiplicand	INT	0	
2	VAR	multiplicator	INT	0	
3	VAR	result	INT	0	

This example uses variables. You may also use constants for the input variables

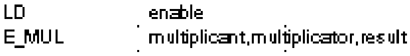
Body

If *enable* is set (TRUE), the *multiplicant* is multiplied with the *multiplicator.* The result will be written in *result*.

LD



IL



```
LD          enable
E_MUL       multiplicant,multiplicator,result
```

E_DIV Divide

Description E_DIV divides the value of the first input variable by the value of the second. E_DIV operates just as the standard operator DIV (see Online Help).

Data types

Data type	I/O	Function
INT, DINT, REAL	1st input	dividend
INT, DINT, REAL	2nd input	divisor
INT, DINT, REAL	output as input	result

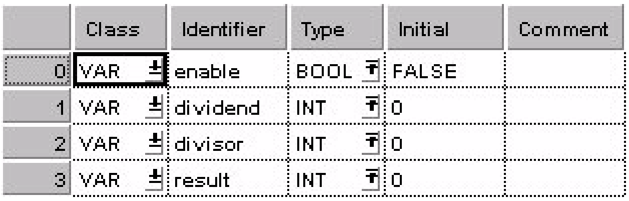
The input variables have to be of the same data type.

- **Only the FP0 can process the data type REAL.**
- **With FP1–C14/C16 E_DIV cannot be used for a 32–bit division (DINT) as this will cause a compiler error.**

Example In this example the function E_DIV is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



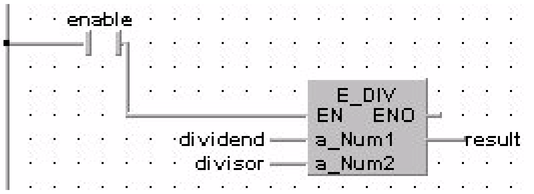
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	dividend	INT	0	
2	VAR	divisor	INT	0	
3	VAR	result	INT	0	

This example uses variables. You may also use constants for the input variables.

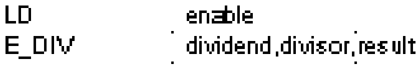
Body

If *enable* is set (TRUE), *dividend* is divided by *divisor.* The result is written in *result.*

LD



IL



```
LD        enable
E_DIV     dividend,divisor,result
```

(E_)MOD

Modular arithmetic division, remainder stored in output variable

Description MOD divides the value of the first input variable by the value of the second. The rest of the integral division (5 : 2 : 2 + rest = 1) is written into the output variable.The remainder of the integral division is written in the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

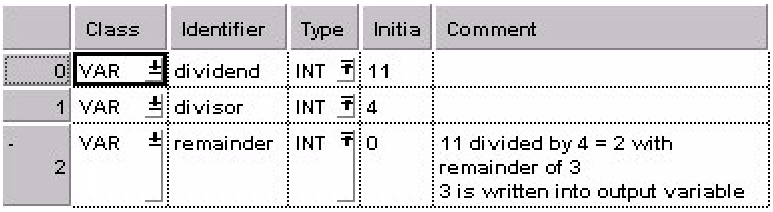
Data type	I/O	Function
INT, DINT	1st input	dividend
INT, DINT	2nd input	divisor
INT, DINT	output as input	remainder

With FP1–C14/C16 E_DIV cannot be used for a 32–bit division (DINT) as this will cause a compiler error.

Example In this example the function MOD is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

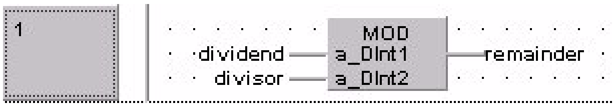


	Class	Identifier	Type	Initia	Comment
0	VAR	dividend	INT	11	
1	VAR	divisor	INT	4	
2	VAR	remainder	INT	0	11 divided by 4 = 2 with remainder of 3 3 is written into output variable

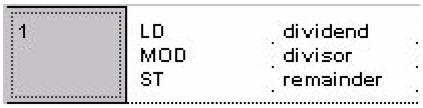
Body

This example uses variables. You may also use constants for the input variables. *Dividend* (11) is divided by *divisor* (4)*.* The remainder (3) of the division is written in *remainder.*

LD



IL



```
1   LD      dividend
    MOD     divisor
    ST      remainder
```

(E_)SQRT Square root

Description SQRT calculates the square root of an input variable of the data type REAL (value ≥ 0.0). The result is written into the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

This function is only available for the FP0.

Data types

Data type	I/O	Function
REAL	input	input value
REAL	output as input	square root of input value

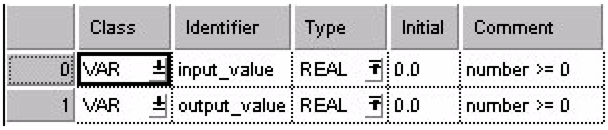
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not ≥ 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



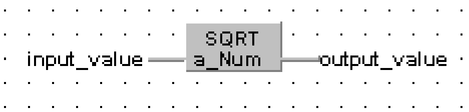
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number >= 0
1	VAR	output_value	REAL	0.0	number >= 0

This example uses variables. You may also use a constant for the input variable.

Body

The square root of *input_value* is calculated and written into *output_value.*

LD



ST

```
output_value:= SQRT(input_value);
```

(E_)SIN Sine

Description SIN calculates the sine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend entering angle data in radians ≥ –2π and ≤ 2π.**
- **This function is only available for the FP0.**

Data types

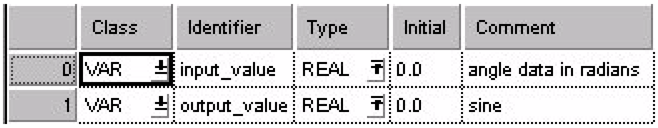
Data type	I/O	Function
REAL	input	input value
REAL	output as input	SINE of input value

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function SIN is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

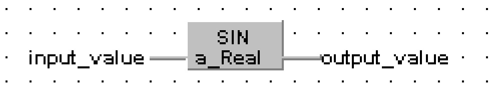


	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	angle data in radians
1	VAR	output_value	REAL	0.0	sine

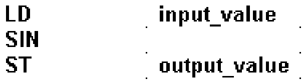
This example uses variables. You may also use a constant for the input variable.

Body The sine of *input_value* is calculated and written into *output_value*.

LD



IL



```
LD      input_value
SIN
ST      output_value
```

(E_)ASIN Arcsine

Description ASIN calculates the arcsine of the input variable and writes the angle data in radians into the output variable. The function returns a value from –π/2 to π/2.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

Data types

Data type	I/O	Function
REAL	input	input value between –1 and +1
REAL	output as input	arcsine of input value in radians

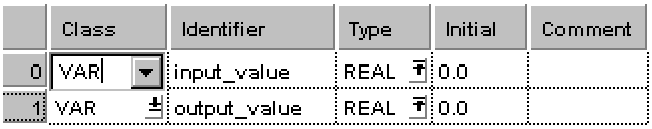
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not ≥ –1.0 and ≤ 1.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



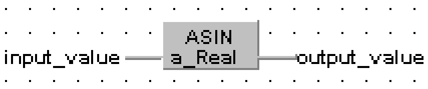
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	
1	VAR	output_value	REAL	0.0	

This example uses variables. You may also use a constant for the input variable.

Body

The arc sine of *input_value* is calculated and written into *output_value*.

LD



ST

```
output_value:=ASIN(input_value);
```

(E_)COS Cosine

Description COS calculates the cosine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value <52707176).

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ –2π and ≤ 2π.**
- **This function is only available for the FP0.**

Data types

Data type	I/O	Function
REAL	input	input value, angle data in radians
REAL	output as input	cosine of input value

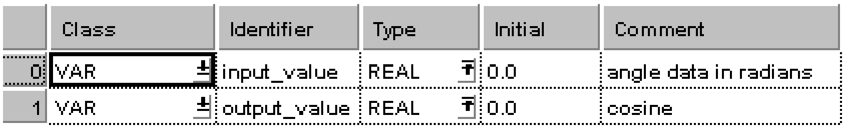
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function COS is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



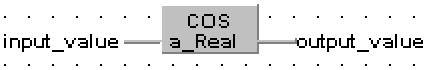
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	angle data in radians
1	VAR	output_value	REAL	0.0	cosine

This example uses variables. You may also use a constant for the input variable.

Body

The cosine of *input_value* is calculated and written into *output_value*.

LD



ST

```
output_value:=COS(input_value);
```

(E_)ACOS Arccosine

Description ACOS calculates the arccosine of the input variable and writes the angle data in radians into the output variable. The function returns a value from 0.0 to π.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

Data types

Data type	I/O	Function
REAL	input	input value between –1 and +1
REAL	output as input	arccosine of input value in radians

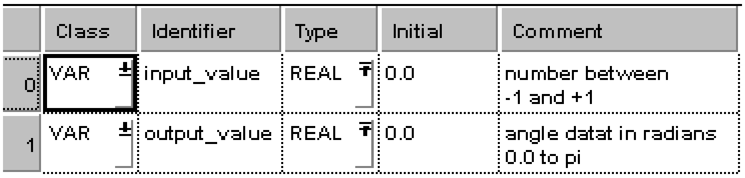
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not ≥ –1.0 and ≤ 1.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



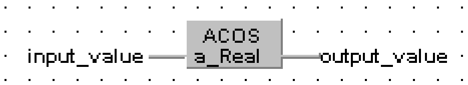
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number between -1 and +1
1	VAR	output_value	REAL	0.0	angle datat in radians 0.0 to pi

This example uses variables. You may also use a constant for the input variable.

Body

The arc cosine of *input_value* is calculated and written into *output_value*.

LD



ST

```
output_value:=ACOS(input_value);
```

(E_)TAN Tangent

Description TAN calculates the tangent of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞

- **The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ –2π and ≤ 2π.**
- **This function is only available for the FP0.**

Data types

Data type	I/O	Function
REAL	input	input value in radians
REAL	output as input	tangent of input value

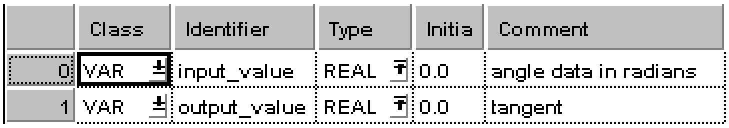
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function TAN is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



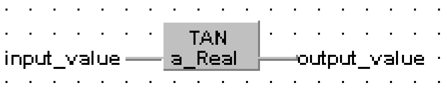
	Class	Identifier	Type	Initia	Comment
0	VAR	input_value	REAL	0.0	angle data in radians
1	VAR	output_value	REAL	0.0	tangent

This example uses variables. You may also use a constant for the input variable.

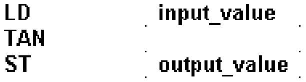
Body

The tangent of *input_value* is calculated and written into *output_value*.

LD



IL



```
LD      input_value
TAN
ST      output_value
```

(E_)ATAN Arctangent

Description ATAN calculates the arctangent of the input variable (value ± 52707176) and writes the angle data in radians into the output variable. The function returns a value greater than –π/2 and smaller than π/2.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

Data types

Data type	I/O	Function
REAL	input	input value between –52707176 and +52707176
REAL	output as input	arctangent of input value in radians

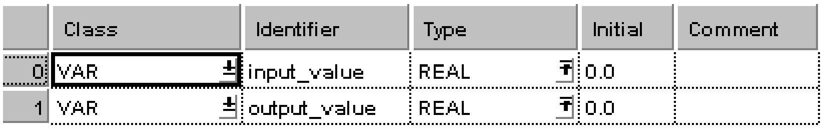
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not ≥ 52707176
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



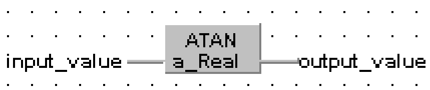
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	
1	VAR	output_value	REAL	0.0	

This example uses variables. You may also use a constant for the input variable.

Body

The arc tangent of *input_value* is calculated and written into *output_value*.

LD



ST

```
output_value:=ATAN(input_value);
```

(E_)LN Natural logarithm

Description LN calculates the logarithm of the input variable (value > 0.0) to the base **e** (Euler's number = 2.7182818) and writes the result into the output variable. This function is the reverse of the EXP function.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

This function is only available for the FP0.

Data types

Data type	I/O	Function
REAL	input	input value
REAL	output as input	natural logarithm of input value

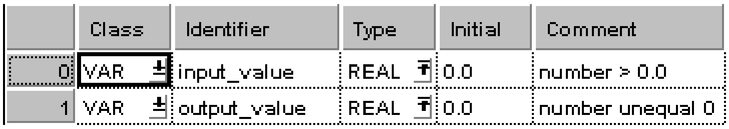
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not > 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function LN is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



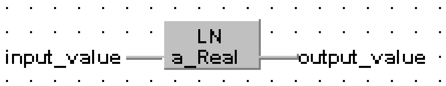
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number > 0.0
1	VAR	output_value	REAL	0.0	number unequal 0

This example uses variables. You may also use a constant for the input variable.

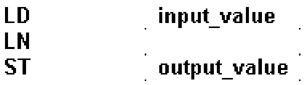
Body

The logarithm of *input_value* is calculated to the base **e** and written into *output_value*.

LD



IL



```
LD      input_value
LN
ST      output_value
```

(E_)LOG — Logarithm

Description LOG calculates the logarithm of the input variable (value > 0.0) to the base 10 and writes the result into the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

This function is only available for the FP0.

Data types

Data type	I/O	Function
REAL	input	input value
REAL	output as input	logarithm of input value

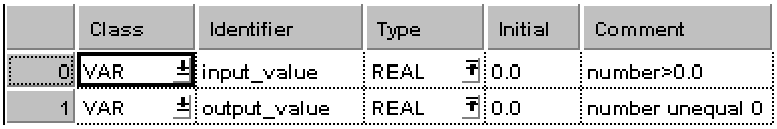
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not > 0.0
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example In this example the function LOG is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



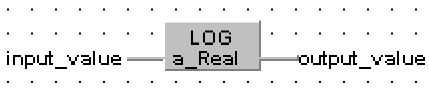
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value	REAL	0.0	number>0.0
1	VAR	output_value	REAL	0.0	number unequal 0

This example uses variables. You may also use a constant for the input variable.

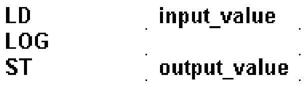
Body

The logarithm of *input_value* is calculated to the base 10 and written into *output_value*.

LD



IL



```
LD      input_value
LOG
ST      output_value
```

(E_)EXP

Exponent of input variable to base e

Description

EXP calculates the power of the input variable to the base **e** (Euler’s number = 2.7182818) and writes the result into the output variable. The input variable has to be greater than –87.33 and smaller than 88.72. This function is the reverse of the LN function.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

This function is only available for the FP0.

Data types

Data type	I/O	Function
REAL	input	input value between –87.33 and +88.72
REAL	output as input	exponent of input variable to base e

Error flags

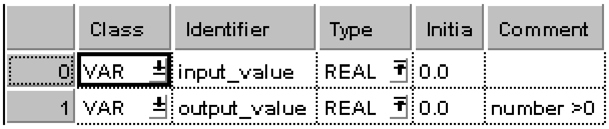
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– input variable does not have the data type REAL or input variable is not > –87.33 and < 88.72
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example

In this example the function EXP is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



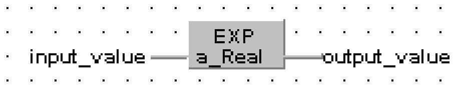
	Class	Identifier	Type	Initia	Comment
0	VAR	input_value	REAL	0.0	
1	VAR	output_value	REAL	0.0	number >0

This example uses variables. You may also use a constant for the input variable.

Body

The power of *input_value* is calculated to the base **e** and written into *output_value*.

LD



IL

```
LD      input_value
EXP
ST      output_value
```

(E_)EXPT

Raises 1st input variable by the power of the 2nd input variable

Description

EXPT raises the first input variable to the power of the second input variable (OUT = IN1^{IN2}) and writes the result into the output variable. Input variables have to be within the range –1.70141 x 10^{38} to 1.70141 x 10^{38}.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

Data types

Data type	I/O	Function
REAL	1st input	input value
REAL	2nd input	exponent of the input value
REAL	output as 1st input	result

Error flags

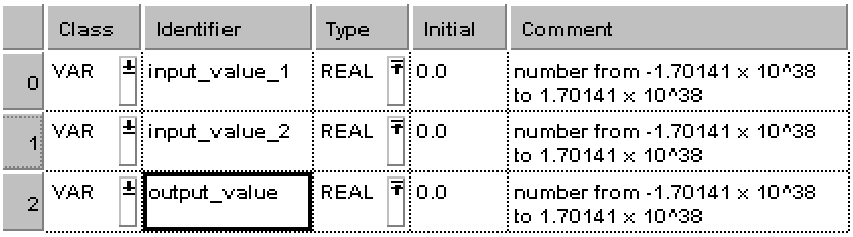
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– first and the second input variable do not have the data type REAL
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	permanently	– output variable is zero
R9009	%MX0.900.9	for an instant	– processing result overflows the output variable

Example

In this example the function EXPT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



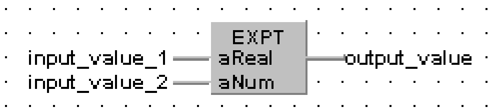
	Class	Identifier	Type	Initial	Comment
0	VAR	input_value_1	REAL	0.0	number from -1.70141 x 10^38 to 1.70141 x 10^38
1	VAR	input_value_2	REAL	0.0	number from -1.70141 x 10^38 to 1.70141 x 10^38
2	VAR	output_value	REAL	0.0	number from -1.70141 x 10^38 to 1.70141 x 10^38

In this example the input variables have been declared. Instead, you may enter constants directly at the input contacts of the function.

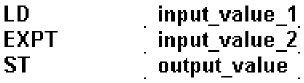
Body

input_value_1 is raised to the power of *input_value_2*. The result is written into *output_value*.

LD



IL



```
LD      input_value_1
EXPT    input_value_2
ST      output_value
```

Chapter 5

Process Data Type Functions

(E_)ADD_TIME Add TIME

Description ADD_TIME adds the times of the two input variables and writes the sum in the output variable.

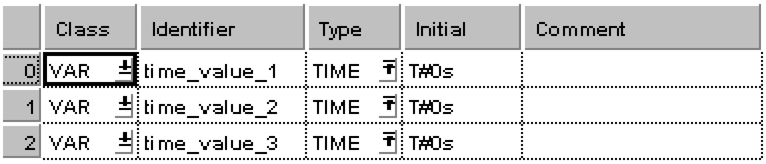
For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
TIME	1st input	augend
TIME	2nd input	addend
TIME	output	sum

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

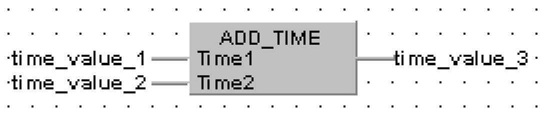


	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#0s	
1	VAR	time_value_2	TIME	T#0s	
2	VAR	time_value_3	TIME	T#0s	

In this example the input variables (*time_value_1* and *time_value_2)* have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body *time_value_1* and *time_value_2* are added. The result is written in *time_value_3*.

LD



ST

```
time_value_3:=ADD_TIME(time_value_1, time_value_2);
```

(E_)SUB_TIME Subtract TIME

Description SUB_TIME subtracts the value of the second input variable from the value of the first and writes the result into the output variable.

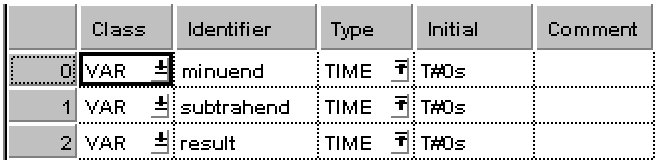
For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

Data type	I/O	Function
TIME	1st input	minuend
TIME	2nd input	subtrahend
TIME	output	result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

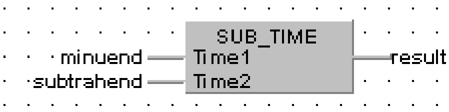


	Class	Identifier	Type	Initial	Comment
0	VAR	minuend	TIME	T#0s	
1	VAR	subtrahend	TIME	T#0s	
2	VAR	result	TIME	T#0s	

In this example the input variables (*minuend* and *subtrahend)* have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body *Subtrahend* is subtracted from *minuend*. The result will be written in *result*.

LD



ST

```
result:= SUB_TIME(minuend, subtrahend);
```

(E_)MUL_TIME_INT Multiply TIME by INTEGER

Description

MUL_TIME_INT multiplies the values of the two input variables with each other and writes the result into the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

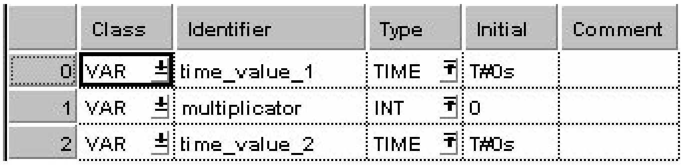
Data type	I/O	Function
TIME	1st input	multiplicand
INT	2nd input	multiplicator
TIME	output	result

Example

In this example the function MUL_TIME_INT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



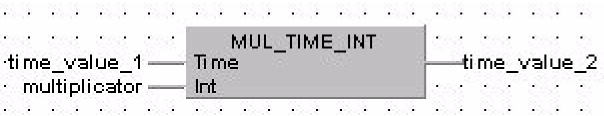
	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#0s	
1	VAR	multiplicator	INT	0	
2	VAR	time_value_2	TIME	T#0s	

In this example the input variables (*time_value_1* and *multiplicator)* have been declared. Instead, you may enter constants directly at the input contacts of a function.

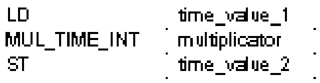
Body

time_value_1 is multiplied with *multiplicator*. The result is written in *time_value_2*.

LD



IL



```
LD              time_value_1
MUL_TIME_INT    multiplicator
ST              time_value_2
```

(E_)MUL_TIME_DINT Multiply TIME by DOUBLE INTEGER

Description MUL_TIME_DINT multiplies the values of the input variables and writes the result to the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

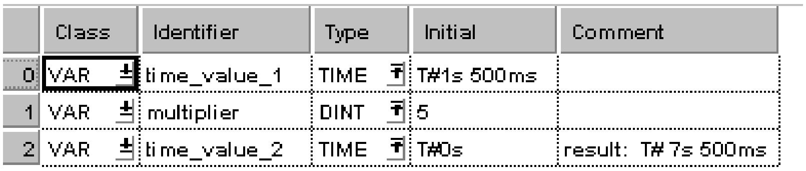
Data types

Data type	I/O	Function
TIME	1st input	dividend
DINT	2nd input	divisor
TIME	output	result

Example In this example the function MUL_TIME_DINT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



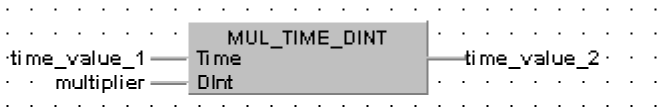
	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#1s 500ms	
1	VAR	multiplier	DINT	5	
2	VAR	time_value_2	TIME	T#0s	result: T# 7s 500ms

In this example, the input variables *time_value* and *multiplier* have been declared. However, you can write a constant directly at the input contact of the function instead.

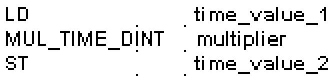
Body

time_value_1 is multiplied by *multiplier*. The result is written in *time_value_2.*

LD



IL



```
LD              time_value_1
MUL_TIME_DINT   multiplier
ST              time_value_2
```

(E_)MUL_TIME_REAL Multiply TIME by REAL

Description MUL_TIME_REAL multiplies the value of the first input variable of the data type TIME by the value of the second input variable of the data type REAL. The REAL value is rounded off to the nearest whole number. The result is written into the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

This function is only available for the FP0.

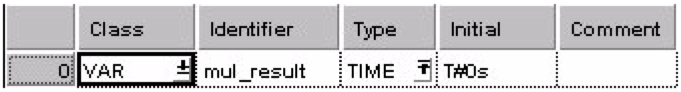
Data types

Data type	I/O	Function
TIME	1st input	multiplicand
REAL	2nd input	multiplicator
TIME	output	result

Example In this example the function MUL_TIME_REAL is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

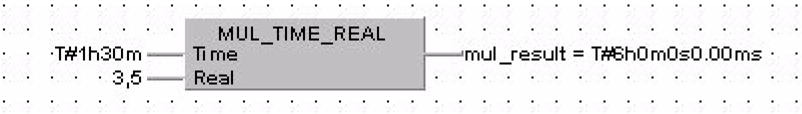


	Class	Identifier	Type	Initial	Comment
0	VAR	mul_result	TIME	T#0s	

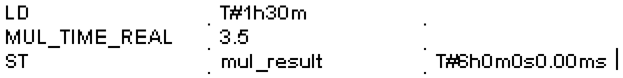
Body

The constant *T#1h30m* is multiplied by the value *3.5*, which is rounded off to *4.0* in the actual calculation. The result is written in *mul_result*. By clicking on the view icon while in the online mode, you can see the result *T#6h0m0s0.00ms* immediately.

LD



IL



```
LD              T#1h30m
MUL_TIME_REAL   3.5
ST              mul_result      T#6h0m0s0.00ms
```

(E_)DIV_TIME_INT Divide TIME by INTEGER

Description DIV_TIME_INT divides the value of the first input variable by the value of the second input variable and writes the result into the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

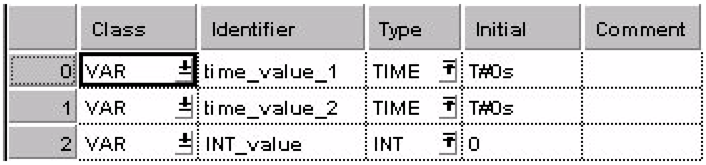
Data types

Data type	I/O	Function
TIME	input	dividend
INT	input	divisor
TIME	output	result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



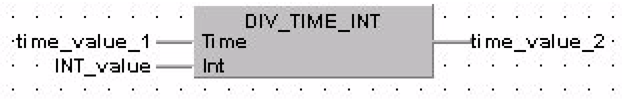
	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#0s	
1	VAR	time_value_2	TIME	T#0s	
2	VAR	INT_value	INT	0	

In this example the input variables (*time_value_1* and *INT_value)* have been declared. Instead, you may enter constants directly at the input contacts of a function.

Body

Time_value_1 is divided by *INT_value*. The result is written in *time_value_2*.

LD



ST

```
time_value_2:=DIV_TIME_INT(time_value_1, INT_value);
```

(E_)DIV_TIME_DINT Divide TIME by DOUBLE INTEGER

Description DIV_TIME_DINT divides the value of the first input variable by the value of the second and writes the result into the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

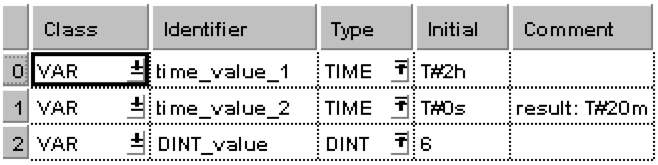
Data types

Data type	I/O	Function
TIME	1st input	dividend
DINT	2nd input	divisor
TIME	output	result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



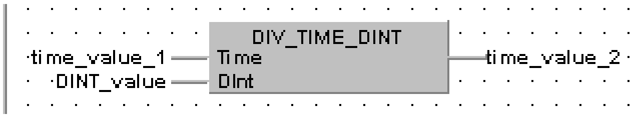
	Class	Identifier	Type	Initial	Comment
0	VAR	time_value_1	TIME	T#2h	
1	VAR	time_value_2	TIME	T#0s	result: T#20m
2	VAR	DINT_value	DINT	6	

In this example, the input variables *time_value_1* and *DINT_value* have been declared. However, you can write a constant directly at the input contact of the function instead.

Body

time_value_1 is divided by *DINT_value*. The result is written in *time_value_2*.

LD



ST

```
time_value_2:=DIV_TIME_DINT(time_value_1, INT_value);
```

(E_)DIV_TIME_REAL Divide TIME by REAL

Description DIV_TIME_REAL divides the value of the first input variable of the data type TIME by the value of the second input variable of the data type REAL. The REAL value is rounded off to the nearest whole number. The result is written into the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is only available for the FP0.**

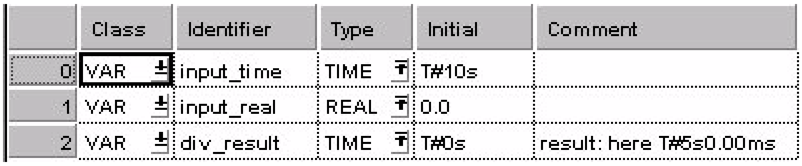
Data types

Data type	I/O	Function
TIME	input	dividend
REAL	input	divisor
TIME	output	result

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

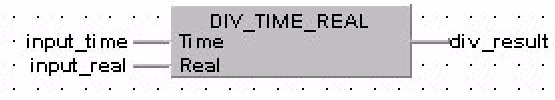


	Class	Identifier	Type	Initial	Comment
0	VAR	input_time	TIME	T#10s	
1	VAR	input_real	REAL	0.0	
2	VAR	div_result	TIME	T#0s	result: here T#5s0.00ms

Body

The value of variable *input_time* is divided by the value of the variable *input_real*. The result is written in *div_result*. In this example the input variables have been declared in the POU header. However, you may enter constants directly at the contact pins of the function.

LD



ST

```
div_result:=DIV_TIME_REAL(input_time, input_real);
```

Chapter 6

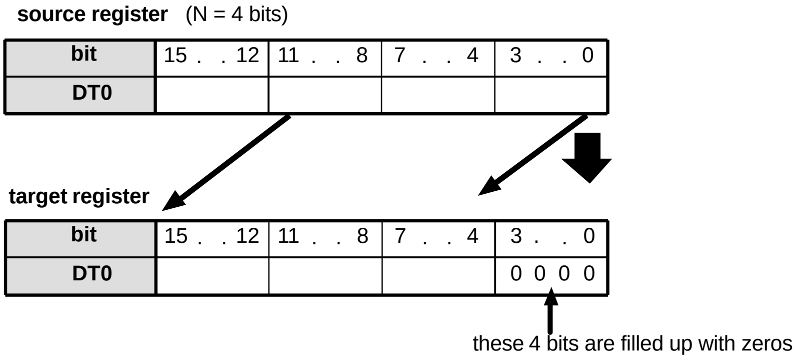
Bitshift Functions

(E_)SHL

Shift bits to the left

Description

SHL shifts a bit value by a defined number of positions (N) to the left and fills the vacant positions with zeros.



source register (N = 4 bits)

bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0				

target register

bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0				0 0 0 0

these 4 bits are filled up with zeros

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

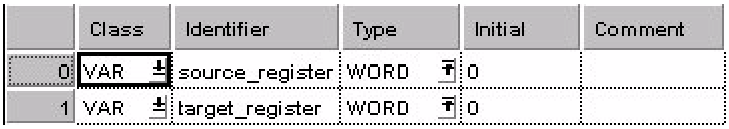
Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is shifted to the left
BOOL, WORD, DWORD	output as input	result

Example

In this example the function SHL is programmed in ladder diagram (LD).

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



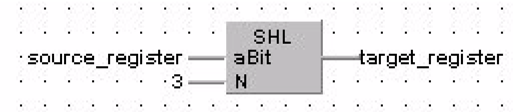
	Class	Identifier	Type	Initial	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

The value for *source_register* are shifted N (3) bits to the left. The resulting vacant bits are filled with zeros. The result is written in *target_register*.

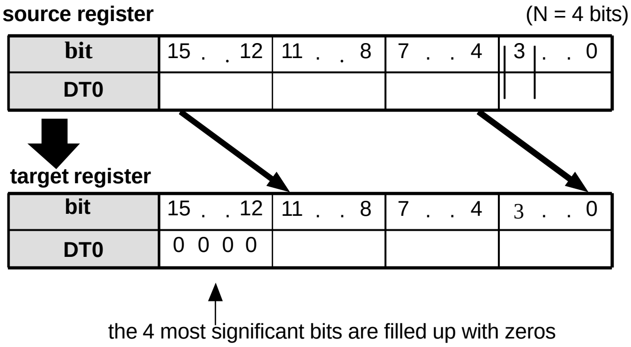
LD



(E_)SHR

Shift bits to the right

Description SHR shifts a bit value by a defined number of positions (N) to the right and fills the vacant positions with zeros.



For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

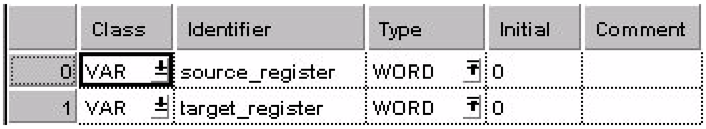
Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is shifted to the right
BOOL, WORD, DWORD	output as input	result

If the second input variable N **(the number of bits to be shifted) is of the data type DWORD, then only the lower 16 bits are taken into account.**

Example In this example the function SHR is programmed in instruction list (IL).

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



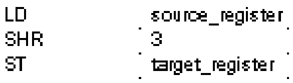
	Class	Identifier	Type	Initial	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

Body

The value for *source_register* are shifted N (3) bits to the right. The resulting vacant bits are filled with zeros. The result is written in *target_register*.

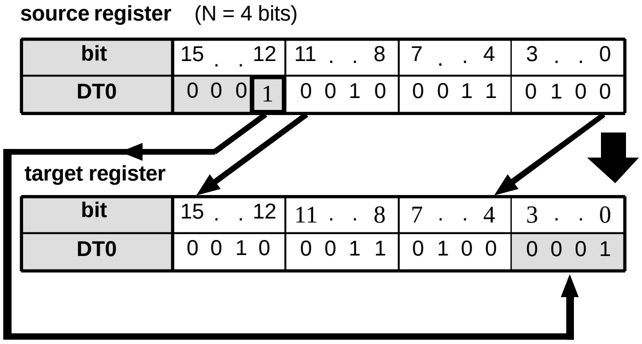
IL



```
LD      source_register
SHR     3
ST      target_register
```

(E_)ROL Rotate bits to the left

Description ROL rotates a defined number (N) of bits to the left.



source register (N = 4 bits)

bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0

target register

bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 0 1 0	0 0 1 1	0 1 0 0	0 0 0 1

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

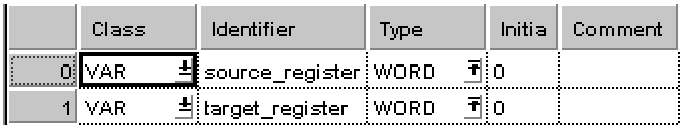
Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is rotated to the left
BOOL, WORD, DWORD	output as input	result

Example In this example the function ROL is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



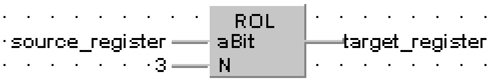
	Class	Identifier	Type	Initia	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

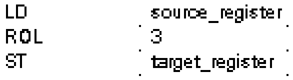
Body

The last N bits (here 3) of *source_register* are left–rotated. The result will be written in *target_register*. This example uses variables. You may also use constants/variables.

LD



IL

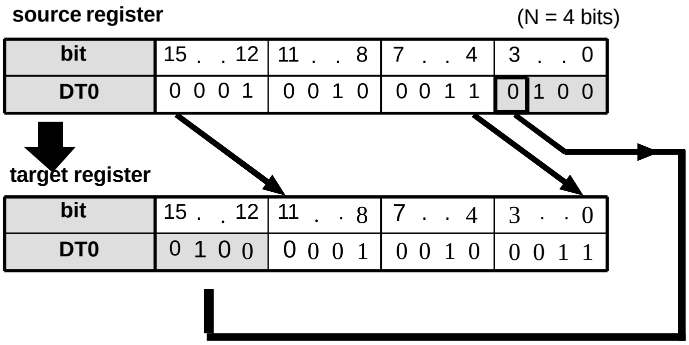


```
LD      source_register
ROL     3
ST      target_register
```

(E_)ROR

Rotate bits to the right

Description ROR rotates a defined number (N) of bits to the right.



bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0

bit	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT0	0 1 0 0	0 0 0 1	0 0 1 0	0 0 1 1

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

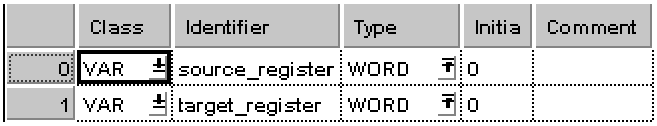
Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	input value
BOOL, WORD, DWORD	2nd input	number of bits by which the input value is rotated to the right
BOOL, WORD, DWORD	output as input	result

Example In this example the function ROR is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



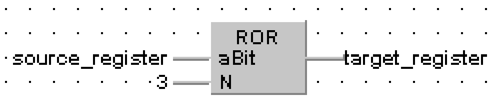
	Class	Identifier	Type	Initia	Comment
0	VAR	source_register	WORD	0	
1	VAR	target_register	WORD	0	

This example uses variables. You may also use a constant for the input variable.

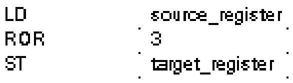
Body

The first N bits (here N = 3) of *source_register* are right–rotated. The result will be written in *target_register*.

LD



IL



```
LD      source_register
ROR     3
ST      target_register
```

Chapter 7

Bitwise Boolean Functions

(E_)AND

Logical AND operation

Description

The content of the accumulator is connected with the operand defined in the operand field by a logical AND operation. The result is transferred to the accumulator.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function AND as standard operator” in the Online Help.

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical AND operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

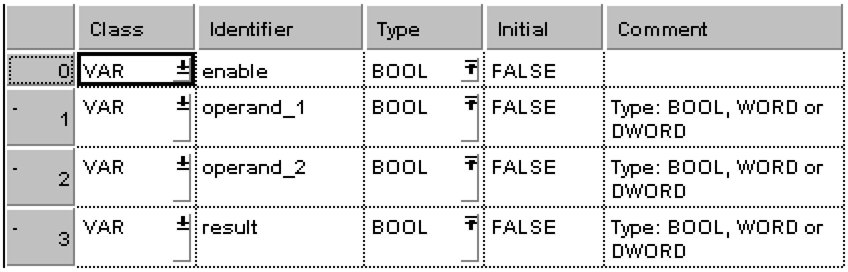
The number of input contacts a_BitN lies in the range of 2 to 28. All operands must be of the same data type.

Example

In this example the function E_AND is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

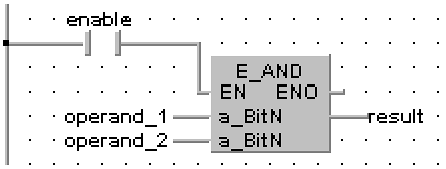


	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	operand_1	BOOL	FALSE	Type: BOOL, WORD or DWORD
2	VAR	operand_2	BOOL	FALSE	Type: BOOL, WORD or DWORD
3	VAR	result	BOOL	FALSE	Type: BOOL, WORD or DWORD

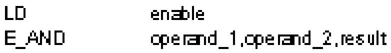
Body

If *enable* is set (TRUE), *operand_1* will be logically AND–linked with *operand_2*. The result will be written into the output variable *result*.

LD



IL



```
LD       enable
E_AND    operand_1,operand_2,result
```

E_OR

Logical OR operation

Description

The content of the accumulator is connected with the operand defined in the operand field by a logical OR operation. The result is transferred to the accumulator.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function OR as standard operator” in the Online Help.

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical OR operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

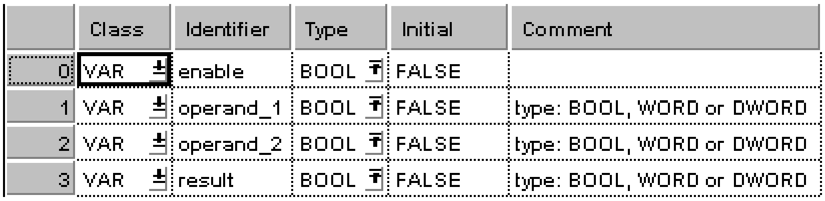
The number of input contacts a_BitN lies in the range of 2 to 28. All operands must be of the same data type.

Example

In this example the function E_OR is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



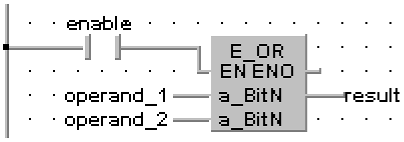
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	operand_1	BOOL	FALSE	type: BOOL, WORD or DWORD
2	VAR	operand_2	BOOL	FALSE	type: BOOL, WORD or DWORD
3	VAR	result	BOOL	FALSE	type: BOOL, WORD or DWORD

In this example the input variables (operand_1,operand_2 and enable) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

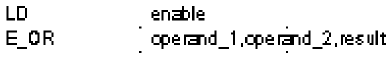
Body

If *enable* is set (TRUE), *operand_1* and *operand_2* are linked with a logical OR. The result will be written in *result*. This example uses variables. You may also use constants for the input variables

LD



IL



```
LD          enable
E_OR        operand_1,operand_2,result
```

E_XOR

Exclusive OR operation

Description

The content of the accumulator is connected with the operand defined in the operand field by a logical XOR operation. The result is transferred to the accumulator.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function XOR as standard operator” in the Online Help.

Data types

Data type	I/O	Function
BOOL, WORD, DWORD	1st input	element 1 of logical XOR operation
BOOL, WORD, DWORD	2nd input	element compared to input 1
BOOL, WORD, DWORD	output as input	result

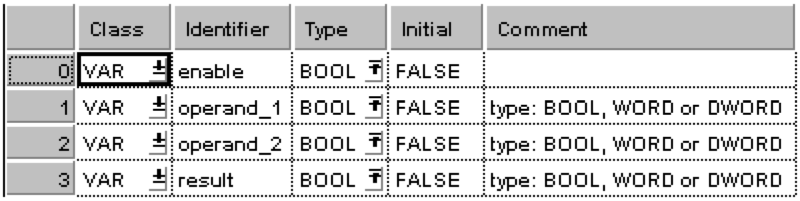
The number of input contacts a_BitN lies in the range of 2 to 28. All operands must be of the same data type.

Example

In this example the function E_XOR is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



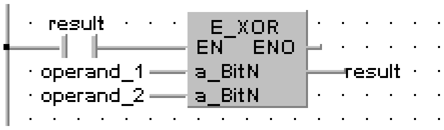
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	operand_1	BOOL	FALSE	type: BOOL, WORD or DWORD
2	VAR	operand_2	BOOL	FALSE	type: BOOL, WORD or DWORD
3	VAR	result	BOOL	FALSE	type: BOOL, WORD or DWORD

In this example the input variables (operand_1,operand_2 and enable) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

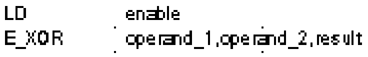
Body

If *enable* is set, the Boolean variables *operand_1* and *operand_2* are logically EXCLUSIVE–OR linked and the result is written in *result*.

LD



IL



```
LD      enable
E_XOR   operand_1,operand_2,result
```

(E_)NOT Bit inversion

Description NOT performs a bit inversion of input variables. The result will be written in the output variable.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

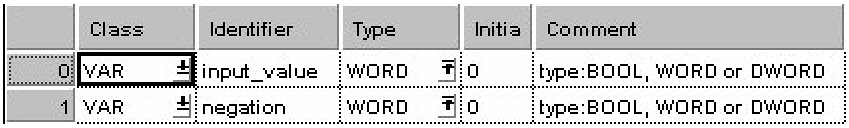
Data type	I/O	Function
BOOL, WORD, DWORD	input	input for NOT operation
BOOL, WORD, DWORD	output as input	result

All operands are of the same data type.

Example In this example the function NOT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



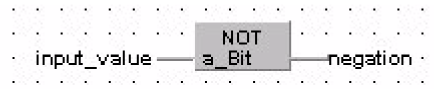
	Class	Identifier	Type	Initia	Comment
0	VAR	input_value	WORD	0	type:BOOL, WORD or DWORD
1	VAR	negation	WORD	0	type:BOOL, WORD or DWORD

This example uses variables. You may also use a constant for the input variable.

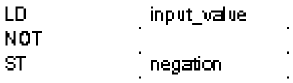
Body

The bits of *input_value* are inversed (0 is inversed to 1 and vice versa). The inversed result is written in *negation.* This example uses variables. You may also use constants for the input variables.

LD



IL



```
LD     input_value
NOT
ST     negation
```

Chapter 8

Selection Functions

(E_)MAX

Maximum value

Description

MAX determines the input variable with the highest value.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

The number of input contacts a_NumN lies in the range of 2 to 28.

Data types

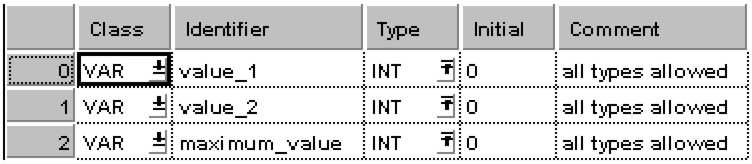
Data type	I/O	Function
all except STRING	1st input	value 1
all except STRING	2nd input	value 2
all except STRING	output as input	result, whichever input variable’s value is greater

Example

In this example the function MAX is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



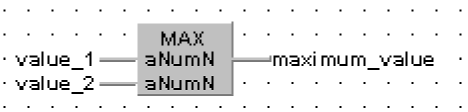
	Class	Identifier	Type	Initial	Comment
0	VAR	value_1	INT	0	all types allowed
1	VAR	value_2	INT	0	all types allowed
2	VAR	maximum_value	INT	0	all types allowed

In this example the input variables (*value_1* and *value_2*) have been declared. Instead, you may enter a constant directly at the input contact of a function.

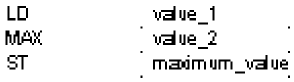
Body

value_1 and *value_2* are compared with each other. The higher value of the two is written in *maximum_value*.

LD



IL



```
LD      value_1
MAX     value_2
ST      maximum_value
```

(E_)MIN

Minimum value

Description

MIN dectects the input variable with the lowest value.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

The number of input contacts a_NumN lies in the range of 2 to 28.

Data types

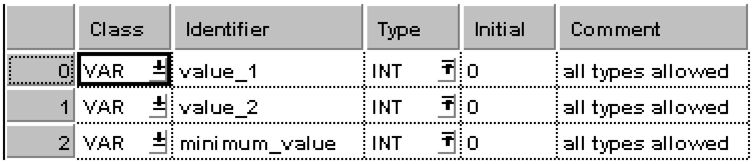
Data type	I/O	Function
all except STRING	1st input	value 1
all except STRING	2nd input	value 2
all except STRING	output as input	result, whichever of the input variable’s value is smallest

Example

In this example the function MIN is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



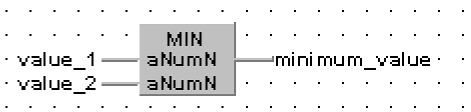
	Class	Identifier	Type	Initial	Comment
0	VAR	value_1	INT	0	all types allowed
1	VAR	value_2	INT	0	all types allowed
2	VAR	minimum_value	INT	0	all types allowed

In this example the input variables (*value_1* and *value_2)* have been declared. Instead, you may enter a constant directly at the input contact of a function.

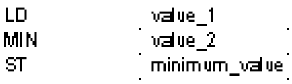
Body

value_1 and *value_2* are compared with each other. The lower value of the two is written in *minimum_value*. This example uses variables. You may also use constants for the input variables.

LD



IL



```
LD      value_1
MIN     value_2
ST      minimum_value
```

(E_)LIMIT

Limit value for input variable

Description

In LIMIT the 1st input variable forms the lower and the 3rd input variable the upper limit value. If the 2nd input variable is within this limit, it will be transferred to the output variable. If it is above this limit, the upper limit value will be transferred, if it is below this limit the lower limit value will be transferred.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

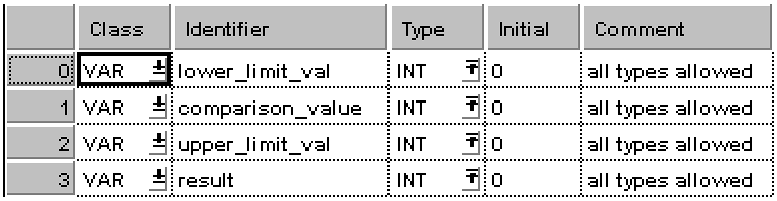
Data type	I/O	Function
all data types	1st input	upper limit
all data types	2nd input	value compared to upper and lower limit
all data types	3rd input	lower limit
all data types	output as input	result, 2nd input value if between upper and lower limit, otherwise the upper or lower limit

Example

In this example the function LIMIT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



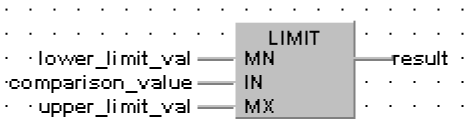
	Class	Identifier	Type	Initial	Comment
0	VAR	lower_limit_val	INT	0	all types allowed
1	VAR	comparison_value	INT	0	all types allowed
2	VAR	upper_limit_val	INT	0	all types allowed
3	VAR	result	INT	0	all types allowed

In this example the input variables (l*ower_limit_val, comparison_value* and *upper_val)* have been declared. Instead, you may enter a constant directly at the input contact of a function.

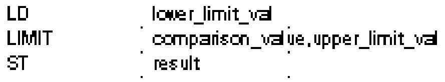
Body

lower_limit_val and *upper_limit_val* form the range where the *comparison_value* has to be, if it has to be transferred to result. If the *comparison_value* is above the *upper_limit_val*, the value of *upper_limit_val* will be transferred to result. If it is below the *lower_limit_val*, the value of *lower_limit_val* will be transferred to result.

LD



IL



```
LD        lower_limit_val
LIMIT     comparison_value,upper_limit_val
ST        result
```

(E_)MUX

Select value from multiple channels

Description

The function Multiplexer selects an input variable and writes its value into the output variable. The 1st input variable determines which input variable is to be written into the output variable. The function MUX can be configured for any desired number of inputs.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

☞

- **The number of input contacts aNumN lies in the range of 2 to 28.**
- **The difference between the functions E_MUX and E_SEL is that in E_MUX you can select between multiple channels with an integer value, while in E_SEL you can only choose between two channels with a Boolean value.**
- **When using the data type STRING, make sure that the length of the result string is equal to or greater than the length of the source string.**

Data types

Data type	I/O	Function
INT	1st input	selects channel for 2nd or 3rd input value to be written to
all data types	2nd input	value 1
all data types	3rd input	value 2
all data types	output as 2nd and 3rd input	result

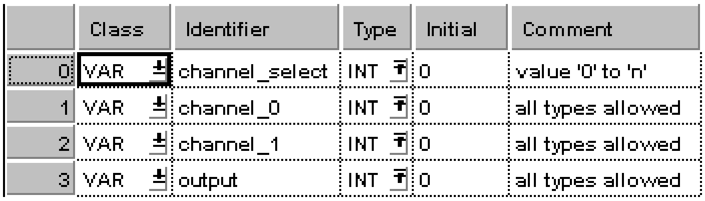
The 2nd and 3rd input variables must be of the same data type.

Example

In this example the function MUX is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



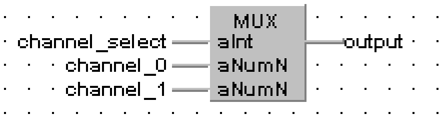
	Class	Identifier	Type	Initial	Comment
0	VAR	channel_select	INT	0	value '0' to 'n'
1	VAR	channel_0	INT	0	all types allowed
2	VAR	channel_1	INT	0	all types allowed
3	VAR	output	INT	0	all types allowed

In this example the input variables (*channel_select, channel_0* and *channel_1)* have been declared. Instead, you may enter a constant directly at the input contact of a function.

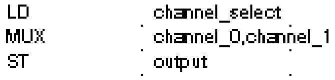
Body

In *channel_select* you find the integer value (0, 1...n) for the selection of *channel_0* or *channel_1*. The result will be written in *output*.

LD



IL



```
LD      channel_select
MUX     channel_0,channel_1
ST      output
```

(E_)SEL Select value from one of two channels

Description

With the first input variable (data type BOOL) of SEL you define which input variable is to be written into the output variable. If the Boolean value = 0 (FALSE), the second input variable will be written into the output variable, otherwise the third.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞

- **The difference between the functions SEL and MUX is that in case of SEL a Boolean value serves for the channel selection, while in case of MUX an integral number (INT) does. Therefore, you can choose between more than two channels with MUX.**
- **When using the data type STRING make sure that the length of the result string is equal to or greater than the length of the source string.**

Data types

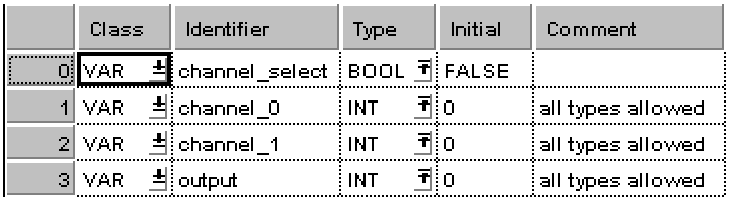
Data type	I/O	Function
BOOL	1st input	selects channel for 2nd or 3rd input value to be written to
all data types	2nd input	value 1
all data types	3rd input	value 2
all data types	output as 2nd and 3rd input	result

Example

In this example the function SEL is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

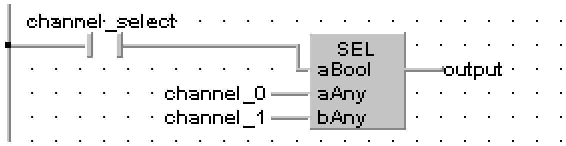


	Class	Identifier	Type	Initial	Comment
0	VAR	channel_select	BOOL	FALSE	
1	VAR	channel_0	INT	0	all types allowed
2	VAR	channel_1	INT	0	all types allowed
3	VAR	output	INT	0	all types allowed

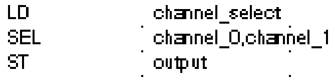
In this example the input variables (*channel_select, channel_0* and *channel_1)* have been declared. Instead, you may enter a constant directly at the input contact of a function.

Body If *channel_select* has the value 0, *channel_0* will be written in output, otherwise *channel_1*. This example uses variables. You may also use constants for the input variables.

LD If *channel_select* has the value 0, *channel_0* will be written into output, otherwise channel_1.



IL



```
LD      channel_select
SEL     channel_0,channel_1
ST      output
```

Chapter 9

Comparison Functions

E_GT Greater than

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater than the reference value, ”TRUE” is stored in the accumulator, otherwise ”FALSE”.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function GT as standard operator” in the Online Help.

The number of input contacts lies in the range of 2 to 28.

Data types

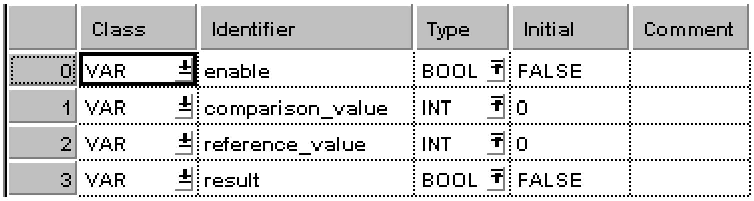
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is greater than reference value

The variables that are compared to each other must be of the same data type.

Example In this example the function E_GT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



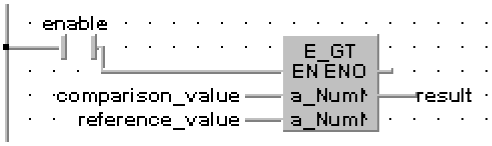
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	comparison_value	INT	0	
2	VAR	reference_value	INT	0	
3	VAR	result	BOOL	FALSE	

Body

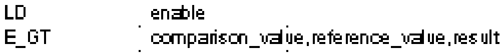
If *enable* is set (TRUE), the *comparison_value* is compared with the *reference_value*. If the *comparison_value* is greater than the *reference_value*, the value TRUE will be written into *result*, otherwise FALSE.

In this example the input variables (*comparison_value, reference_value* and *enable*) have been declared. Instead, you may enter constants directly at the input contacts of a function (enable input e.g. for tests).

LD



IL



```
LD      enable
E_GT    comparison_value,reference_value,result
```

E_GE Greater than or equal to

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater or equal to the reference value, ”TRUE” is stored in the accumulator, otherwise ”FALSE”.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function GE as standard operator” in the Online Help.

The number of input contacts lies in the range of 2 to 28.

Data types

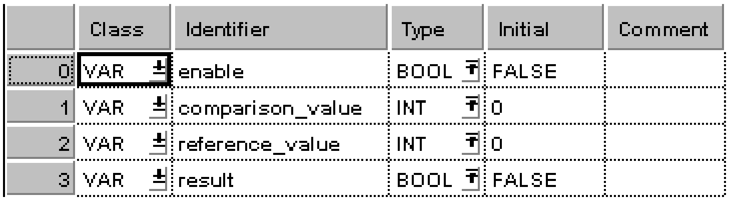
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is greater than or equal to reference value

The variables that are compared to each other must be of the same data type.

Example In this example the function E_GE is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



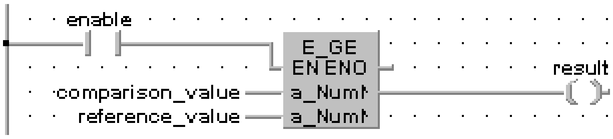
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	comparison_value	INT	0	
2	VAR	reference_value	INT	0	
3	VAR	result	BOOL	FALSE	

Body

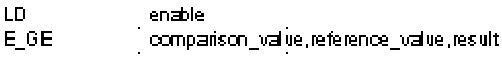
If *enable* is set (TRUE), the *comparison_value* is compared with the *reference_value*. If the *comparison_value* is greater than or equal to the *reference_value*, the value TRUE will be written in *result*, otherwise FALSE.

This example uses variables. You may also use constants for the input variables.

LD



IL



```
LD          enable
E_GE        comparison_value,reference_value,result
```

E_EQ Equal to

Description The content of the accumulator is compared with the operand defined in the operand field. If both values are equal, ”TRUE” is stored in the accumulator, otherwise ”FALSE”.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function EQ as standard operator” in the Online Help.

The number of input contacts lies in the range of 2 to 28.

Data types

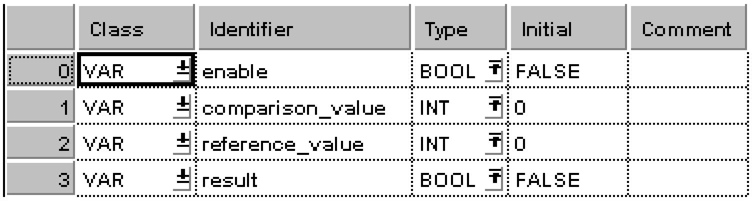
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is equal to reference value

The variables that are compared to each other must be of the same data type.

Example In this example the function E_EQ is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



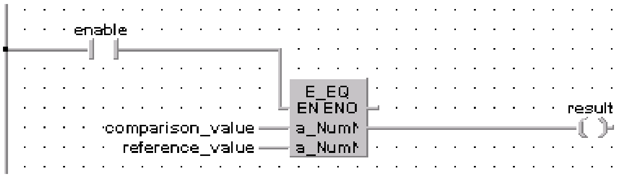
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	comparison_value	INT	0	
2	VAR	reference_value	INT	0	
3	VAR	result	BOOL	FALSE	

Body

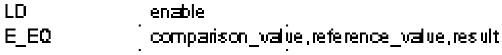
If *enable* is set (TRUE), the variable *comparison_value* is compared with the variable *reference_value*. If the values of the two variables are identical, the value TRUE will be written in *result*, otherwise FALSE.

This example uses variables. You may also use constants for the input variables.

LD



IL



```
LD      enable
E_EQ    comparison_value,reference_value,result
```

E_LE Less than or equal to

Description The content of the accumulator is compared to the operand defined in the operand field. If the accumulator is less than or equal to the reference value, ”TRUE” is stored in the accumulator, otherwise ”FALSE”.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function LE as standard operator” in the Online Help.

The number of input contacts lies in the range of 2 to 28.

Data types

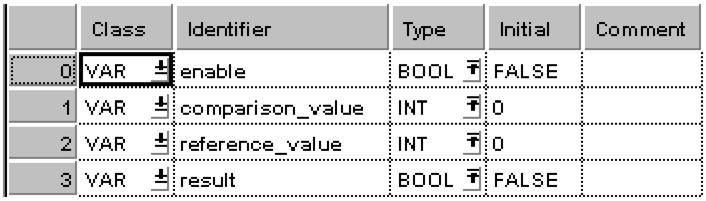
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is less than or equal to the reference value

The variables that are compared to each other must be of the same data type.

Example In this example the function E_LE is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



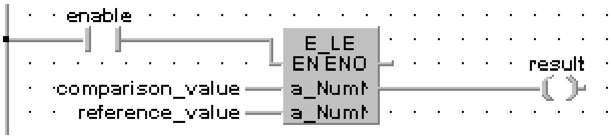
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	comparison_value	INT	0	
2	VAR	reference_value	INT	0	
3	VAR	result	BOOL	FALSE	

Body

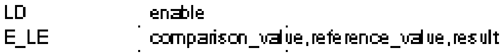
If *enable* is set (TRUE), the *comparison_value* is compared with the variable *reference_value*. If the *comparison_value* is less than or equal to the *reference_value*, TRUE will be written in *result*, otherwise FALSE.

This example uses variables. You may also use constants for the input variables.

LD



IL



```
LD        enable
E_LE      comparison_value,reference_value,result
```

E_LT Less than

Description The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is less than the reference value, ”TRUE” is stored in the accumulator, otherwise ”FALSE”.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function LT as standard operator” in the Online Help.

The number of input contacts lies in the range of 2 to 28.

Data types

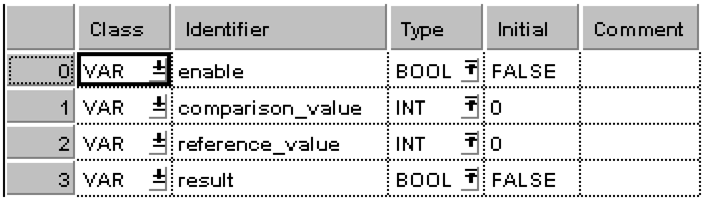
Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is less than the reference value

The variables that are compared to each other must be of the same data type.

Example In this example the function E_LT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



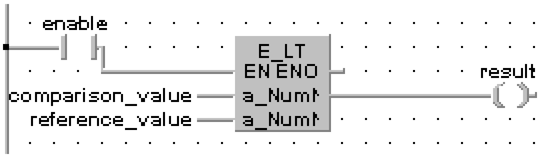
	Class	Identifier	Type	Initial	Comment
0	VAR	enable	BOOL	FALSE	
1	VAR	comparison_value	INT	0	
2	VAR	reference_value	INT	0	
3	VAR	result	BOOL	FALSE	

Body

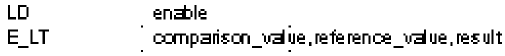
If *enable* is set (TRUE), the *comparison_value* is compared with the *reference_value*. If the *comparison_value* is less than or equal to the *reference_value*, TRUE will be written in *result*, otherwise FALSE.

This example uses variables. You may also use constants for the input variables

LD



IL



```
LD        enable
E_LT      comparison_value,reference_value,result
```

E_NE Not equal

Description The content of the accumulator is compared with the operand defined in the operand field. If the values are not equal, ”TRUE” is stored in the accumulator, otherwise ”FALSE”.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function NE as standard operator” in the Online Help.

☞ **The number of input contacts lies in the range of 2 to 28.**

Data types

Data type	I/O	Function
all data types	1st input	value for comparison
all data types	2nd input	reference value
BOOL	output	result, TRUE if 2nd input value is not equal to the reference value

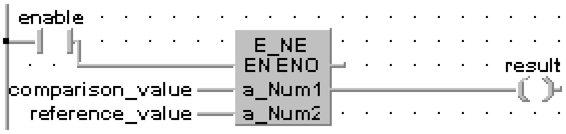
The variables that are compared to each other must be of the same data type.

Example In this example the function E_NE is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

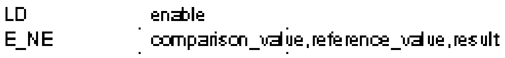
POU header In the POU header, all input and output variables are declared that are used for programming this function.

Body If *enable* is set (TRUE), the *comparison_value* is compared with the *reference_value*. If the two values are unequal, TRUE will be written into *result*, otherwise FALSE. In this example the input variables (*comparison_value, reference_value* and *enable*) have been declared. However, you may enter constants directly into the function (enable input e.g. for tests).

LD



IL



```
LD      enable
E_NE    comparison_value,reference_value,result
```

Chapter 10

Bistable Function Blocks

(E_)SR Set/reset

Description

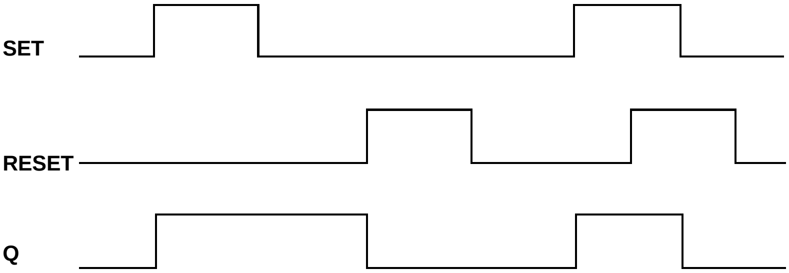
The function block SR (set/reset) or E_SR allows you to both set and reset an output. For the SR you declare the following:

SET: **set**
The output Q is set for each rising edge at SET.

RESET: **reset**
The output Q is reset for each rising edge detected at RESET, except if SET is set (see time chart)

Q: **signal output**
is set if a rising edge is detected at SET; is reset if a rising edge is detected at RESET, and if the SET is not set.

Time Chart



For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the "function with enable" in the Online Help.

☞

- **Q is set if a rising edge is detected at both inputs (Set and Reset)**
- **Upon initialising, Q always has the status zero (reset).**

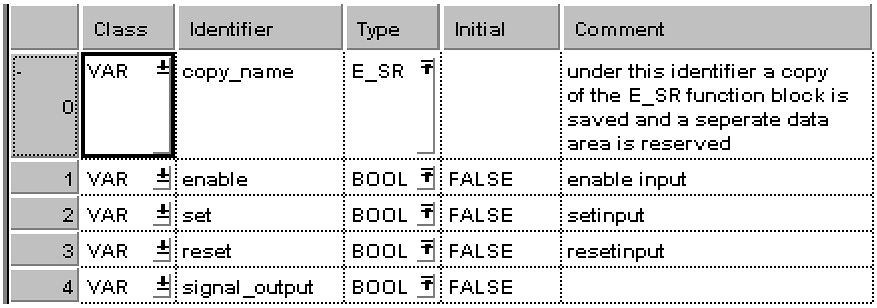
Data types

Data type	I/O	Function
BOOL	1st input	set
BOOL	2nd input	reset
BOOL	output	set or reset depending on inputs

Example In this example the function SR is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*, and a separate data area is reserved.

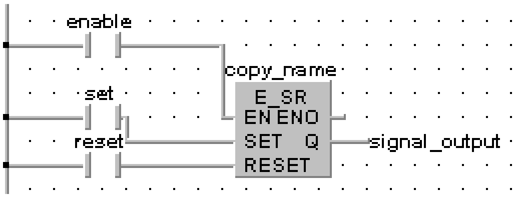


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	E_SR		under this identifier a copy of the E_SR function block is saved and a seperate data area is reserved
1	VAR	enable	BOOL	FALSE	enable input
2	VAR	set	BOOL	FALSE	setinput
3	VAR	reset	BOOL	FALSE	resetinput
4	VAR	signal_output	BOOL	FALSE	

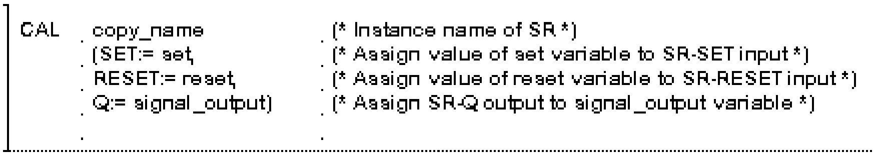
Body

If *set* is set (status = TRUE), *signal_output* will be set. If only *reset* is set, the *signal_output* will be reset (status = FALSE). If both *set* and *reset* are set, *signal_output* will be set.

LD



IL



```
CAL  copy_name            (* Instance name of SR *)
     (SET:= set,          (* Assign value of set variable to SR-SET input *)
     RESET:= reset,       (* Assign value of reset variable to SR-RESET input *)
     Q:= signal_output)   (* Assign SR-Q output to signal_output variable *)
```

The nomination *copy_name.SET* or *copy_name.RESET* etc. has to be maintained in the IL.

(E_)RS Reset/set

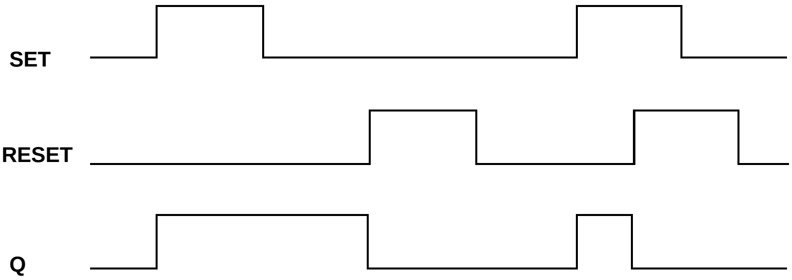
Description The function block RS (reset/set) or E_RS allows you to both reset and set an output. For the RS you declare the following:

SET: **set**
The output Q is set for each rising edge at SET, if RESET is not set.

RESET: **reset**
The output Q is reset for each rising edge at RESET.

Q: **signal output**
is set, if a rising edge is detected at SET and if RESET is not set; is reset, if a rising edge is detected at RESET.

Time Chart



For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **Q is reset if a rising edge is detected at both inputs.**

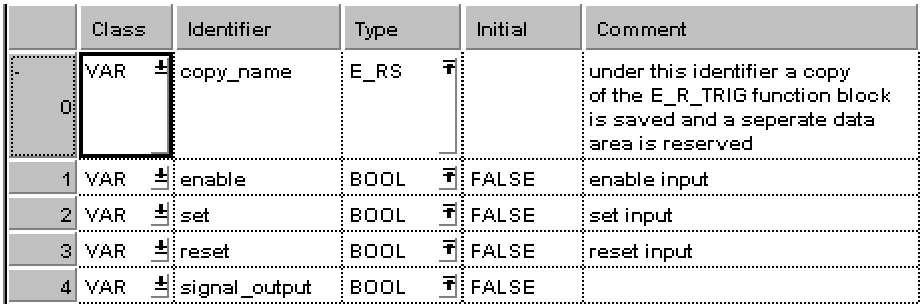
Data types

Data type	I/O	Function
BOOL	1st input	set
BOOL	2nd input	reset
BOOL	output	set or reset depending on inputs

Example In this example the function RS is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*, and a separate data area is reserved.

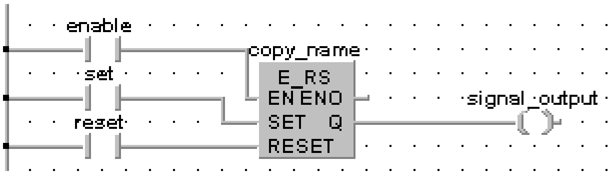


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	E_RS		under this identifier a copy of the E_R_TRIG function block is saved and a seperate data area is reserved
1	VAR	enable	BOOL	FALSE	enable input
2	VAR	set	BOOL	FALSE	set input
3	VAR	reset	BOOL	FALSE	reset input
4	VAR	signal_output	BOOL	FALSE	

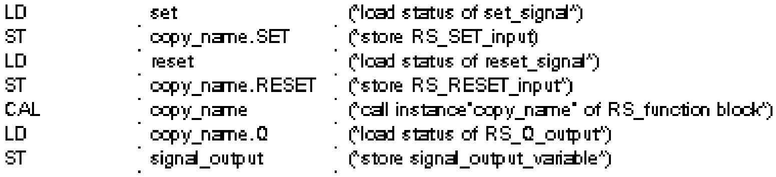
Body

If *set* is set (status = TRUE) the *signal_output* will be set. If only *reset* is set, the *signal_output* will be reset (status = FALSE). If both *set* and *reset* are set, the *signal_output* will be reset to FALSE.

LD



IL



```
LD     set                ("load status of set_signal")
ST     copy_name.SET      ("store RS_SET_input)
LD     reset              ("load status of reset_signal")
ST     copy_name.RESET    ("store RS_RESET_input")
CAL    copy_name          ("call instance"copy_name" of RS_function block")
LD     copy_name.Q        ("load status of RS_Q_output")
ST     signal_output      ("store signal_output_variable")
```

The nomination *copy_name.SET* or *copy_name.RESET* etc. has to be maintained in the IL.

Chapter 11

Edge Detection

(E_)R_TRIG Rising edge trigger

Description The function block R_TRIG (rising edge trigger) or E_R_TRIG allows you to recognize a rising edge at an input. For R_TRIG declare the following:

CLK: signal input
the output Q is set for each rising edge at the signal input (clk = clock)

Q: signal output
is set when a rising edge is detected at CLK.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

The output Q of a function block (E_)R_TRIG remains set for a complete PLC cycle after the occurrence of a rising edge (status change FALSE –> TRUE) at the CLK input and is then reset in the following cycle.

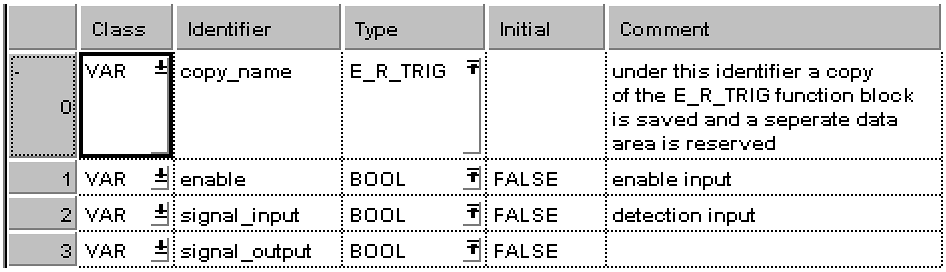
Data types

Data type	I/O	Function
BOOL	input CLK	detects rising edge for clock
BOOL	output Q	set when rising edge detected

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*, and a separate data area is reserved.

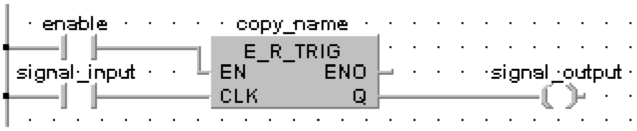


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	E_R_TRIG		under this identifier a copy of the E_R_TRIG function block is saved and a seperate data area is reserved
1	VAR	enable	BOOL	FALSE	enable input
2	VAR	signal_input	BOOL	FALSE	detection input
3	VAR	signal_output	BOOL	FALSE	

Body

signal_output will be set if a rising edge is detected at *signal_input*.

LD



ST

```
copy_name(CLK:= signal_input, Q:= signal_output);
```

(E_)F_TRIG Falling edge trigger

Description The function block F_TRIG (falling edge trigger) or E_F_TRIG allows you to recognize a falling edge at an input. For F_TRIG declare the following:

CLK: **signal input**
the output Q is set for each falling edge at the signal input (clk = clock)

Q: **signal output**
is set if a falling edge is detected at CLK.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

Data types

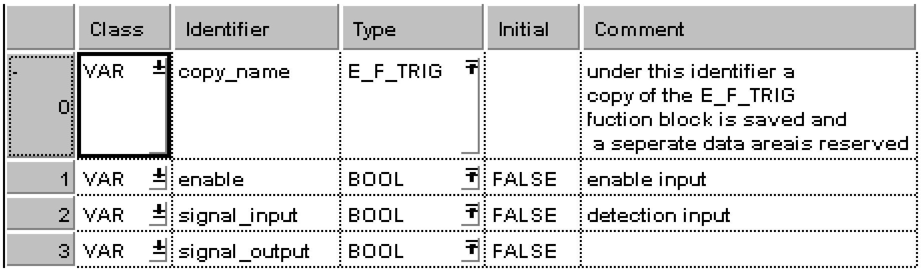
Data type	I/O	Function
BOOL	input CLK	detects falling edge at input clock
BOOL	output Q	is set if falling edge is detected at input

The output Q of a function block (E_)F_TRIG remains set for a complete PLC cycle after the occurrence of a falling edge (status change FALSE –> TRUE) at the CLK input and is then reset in the following cycle.

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

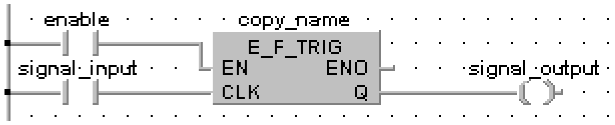
In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*, and a separate data area is reserved.



	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	E_F_TRIG		under this identifier a copy of the E_F_TRIG fuction block is saved and a seperate data areais reserved
1	VAR	enable	BOOL	FALSE	enable input
2	VAR	signal_input	BOOL	FALSE	detection input
3	VAR	signal_output	BOOL	FALSE	

Body *signal_output* will be set if a falling edge is detected at *signal_input*.

LD



ST

```
copy_name(CLK:= signal_input, Q:= signal_output);
```

Chapter 12

Counter

(E_)CTU

Up counter

Description

The function block CTU (count up) allows you to program counting procedures. For CTU declare the following:

CU: **clock generator**
the value 1 is added to CV for each rising edge at CU, except if RESET is set

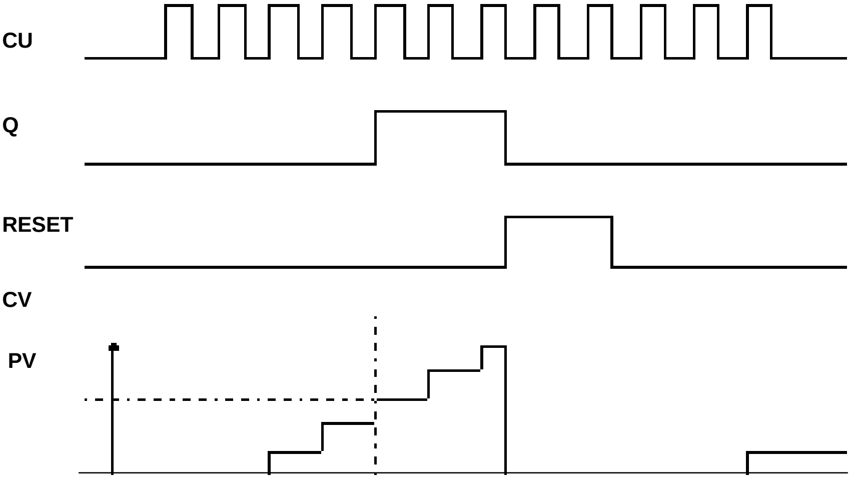
RESET: **reset**
CV is reset to zero for each rising edge at RESET

PV: **set value**
if PV (preset value) is reached, Q is set

Q: **signal output**
is set if CV is greater than/equal to PV

CV: **current value**
contains the addition result (CV = current value)

Time Chart



For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

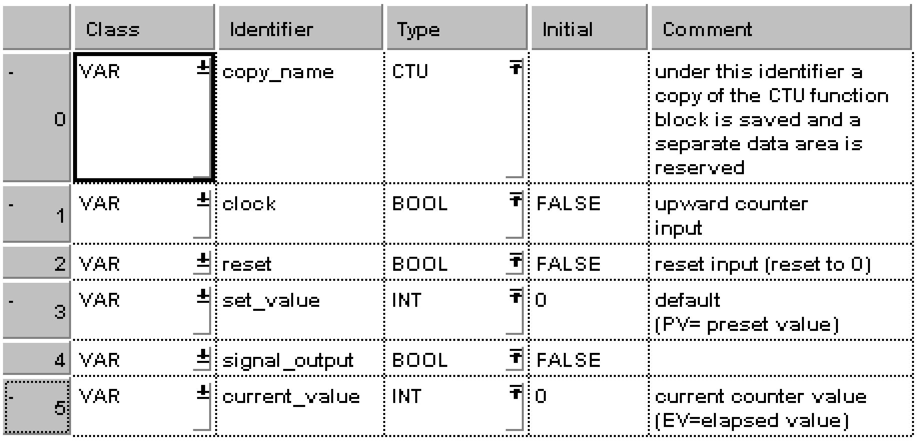
Data types

Data type	I/O	Function
BOOL	input CU	detects rising edge, adds 1 to CV
BOOL	input RESET	resets CV to 0 at rising edge
INT	input PV	set value
BOOL	output Q	set if CV >= PV
INT	output CV	current value

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*. A separate data area is reserved for this copy.

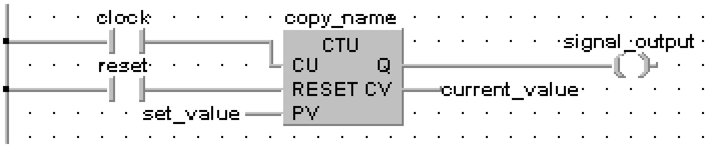


	Class	Identifier	Type	Initial	Comment
- 0	VAR	copy_name	CTU		under this identifier a copy of the CTU function block is saved and a separate data area is reserved
- 1	VAR	clock	BOOL	FALSE	upward counter input
2	VAR	reset	BOOL	FALSE	reset input (reset to 0)
- 3	VAR	set_value	INT	0	default (PV= preset value)
4	VAR	signal_output	BOOL	FALSE	
- 5	VAR	current_value	INT	0	current counter value (EV=elapsed value)

Body

If *reset* is set (status = TRUE), *current_value* (CV) will be reset. If a rising edge is detected at *clock*, the value 1 will be added to *current_value*. If a rising edge is detected at *clock,* this procedure will be repeated until *current_value* is greater than/equal to *set_value*. Then, *signal_output* will be set.

LD



ST

```
copy_name( CU:= clock, RESET:= reset, PV:= set_value, Q:=
signal_output, CV:= current_value);
```

(E_)CTD Down counter

Description The function block CTD (count down) allows you to program counting procedures. For CTD declare the following:

CD: **clock generator input**
the value 1 is subtracted from the current value CV for each rising edge detected at CD, except if LOAD is set or CV has reached the value zero.

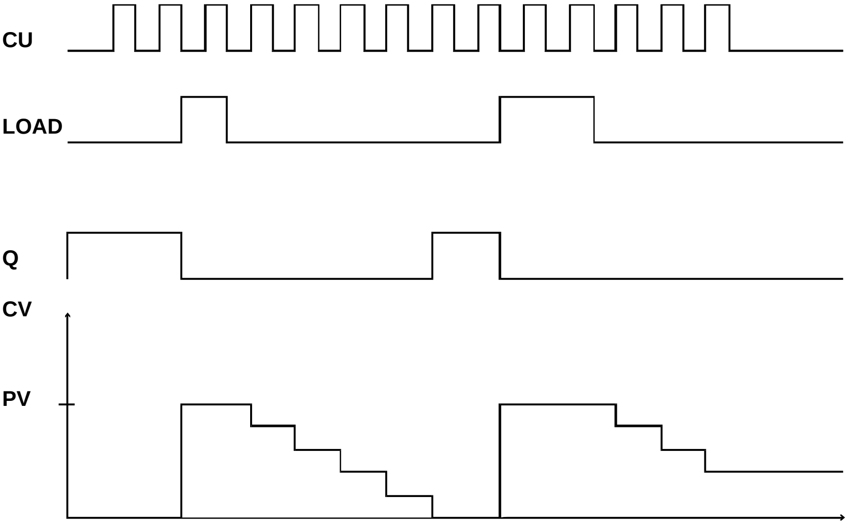
LOAD: **set**
with LOAD the counter state is reset to PV

PV: **preset value**
is the value subjected to subtraction during the first counting procedure

Q: **signal output**
is set if CV = zero

CV: **current value**
contains the current subtraction result (CV = current value)

Time Chart



For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

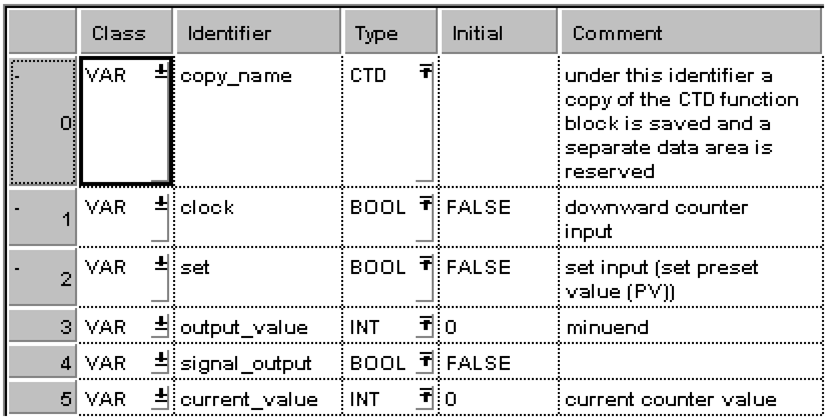
Data types

Data type	I/O	Function
BOOL	input CD	subtracts 1 from CV at rising edge
BOOL	input LOAD	resets counter to PV
INT	input PV	preset value
BOOL	output Q	signal output, set if CV = 0
INT	output CV	current value

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*, and a separate data area is reserved.

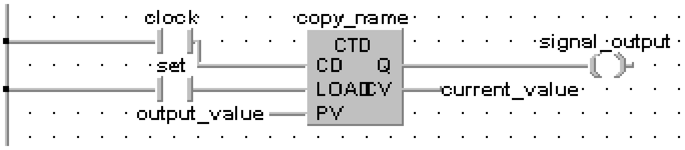


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTD		under this identifier a copy of the CTD function block is saved and a separate data area is reserved
1	VAR	clock	BOOL	FALSE	downward counter input
2	VAR	set	BOOL	FALSE	set input (set preset value (PV))
3	VAR	output_value	INT	0	minuend
4	VAR	signal_output	BOOL	FALSE	
5	VAR	current_value	INT	0	current counter value

Body

If *set* is set (status = TRUE), the *preset_value* (PV) is loaded in the *current_value* (CV). The value 1 will be subtracted from the *current_value* each time a rising edge is detected at *clock.* This procedure will be repeated until the *current_value* is greater than/equal to zero. Then *signal_output* will be set.

LD



ST

```
IF set THEN     (* first cycle *)

      load:=TRUE; (* load has to be TRUE,
            to set current_value to output_value *)
      clock:=FALSE;
END_IF;
copy_name(CD:= clock, LOAD:= set, PV:= output_value, Q:=
signal_output, CV:= current_value);
load:=FALSE;(* now current_value got the right value, load
doesn't need to be *)
            (* TRUE any longer *)
```

(E_)CTUD Up/down counter

Description The function block CTUD (count up/down) allows you to program counting procedures (up and down). For CTUD declare the following:

CU: **count up**
the value 1 is added to the current CV for each rising edge detected at CU, except if RESET and/or LOAD is/are set.

CD: **count down**
the value 1 is subtracted from the current CV for each rising edge detected at CD, except RESET and/or LOAD is/are set and if CU and CD are simultaneously set. In the latter case counting will be upwards.

RESET: **reset**
if RESET is set, CV will be reset

LOAD: **set**
if LOAD is set, PV is loaded to CV. This, however, does not apply if RESET is set simultaneously. In this case, LOAD will be ignored.

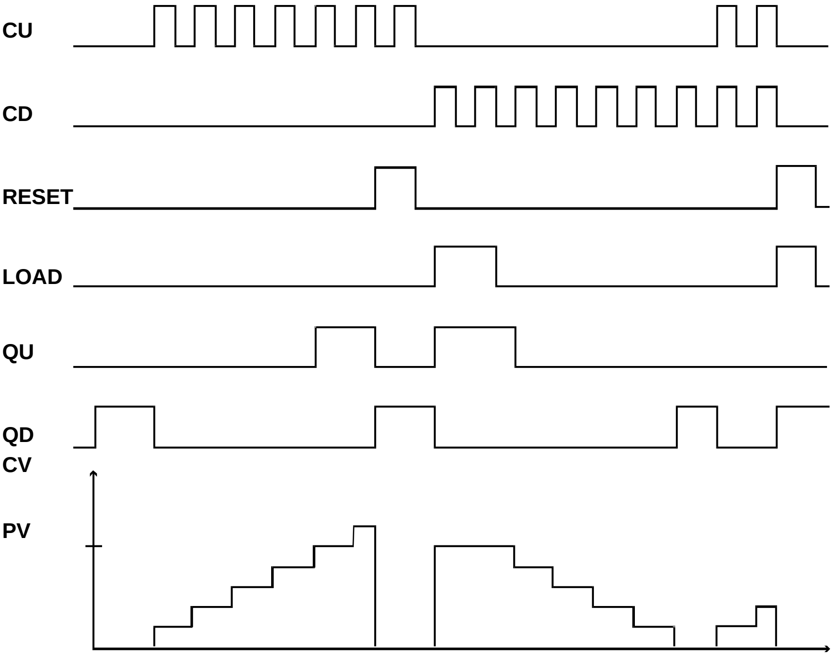
PV: **preset value**
defines the preset value which is to be attained with the addition or subtraction (PV = preset value)

QU: **signal output – count up**
is set if CV is greater than/equal to PV

QD: **signal output – count down**
is set if CV = zero

CV: **current value**
is the addition/subtraction result (CV = current value)

Time Chart



For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

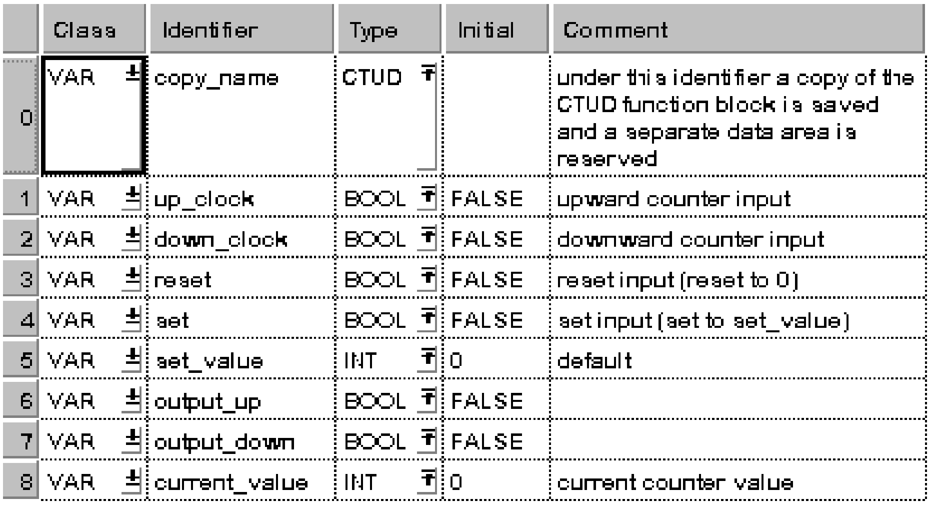
Data types

Data type	I/O	Function
BOOL	input CU	count up
BOOL	input CD	count down
BOOL	input RESET	resets CV if set
BOOL	input LOAD	loads PV to CV
INT	input PV	set value
BOOL	output QU	signal output count up
BOOL	output QD	signal output count down
INT	output CV	current value

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*. A separate data area is reserved for this copy.



	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	CTUD		under this identifier a copy of the CTUD function block is saved and a separate data area is reserved
1	VAR	up_clock	BOOL	FALSE	upward counter input
2	VAR	down_clock	BOOL	FALSE	downward counter input
3	VAR	reset	BOOL	FALSE	reset input (reset to 0)
4	VAR	set	BOOL	FALSE	set input (set to set_value)
5	VAR	set_value	INT	0	default
6	VAR	output_up	BOOL	FALSE	
7	VAR	output_down	BOOL	FALSE	
8	VAR	current_value	INT	0	current counter value

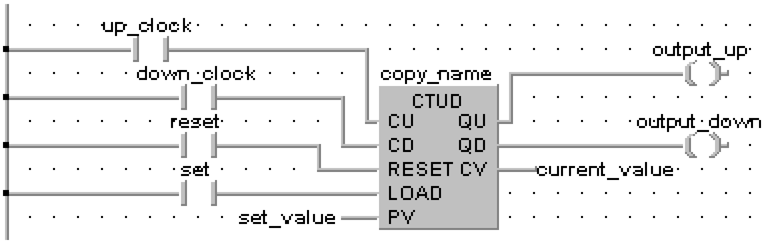
Body

Count up:
If *reset* is set, the *current_value* (CV) will be reset. If up*_clock* is set, the value 1 is added to the *current_value*. This procedure is repeated for each rising edge detected at up*_clock* until the *current value* is greater than/equal to the *set_value*. Then *output_up* is set. The procedure is not conducted, if *reset* and/or *set* is/are set.

Count down:
If *set* is set (status = TRUE), the *set_value* (PV = preset value) will be loaded in

the *current_value* (CV). If *down_clock* is set, the value 1 is subtracted from *set_value* at each clock. This procedure is repeated at each clock until the *current_value* is smaller than/equal to zero. Then,
signal_output is set. The procedure will not be conducted, if *reset* and/or *set* is/are set or if CU and CV are set at the same time. In the latter case, counting will be downwards.

LD



ST

```
copy_name(CU:= up_clock, CD:= down_clock, RESET:= reset,
LOAD:= set, PV:= set_value, QU:= output_up, QD:=
output_down, CV:= current_value);
```

Chapter 13

Timer

(E_)TP Timer with defined period

Description The function block TP allows you to program a clock timer with a defined clock period. For TP declare the following:

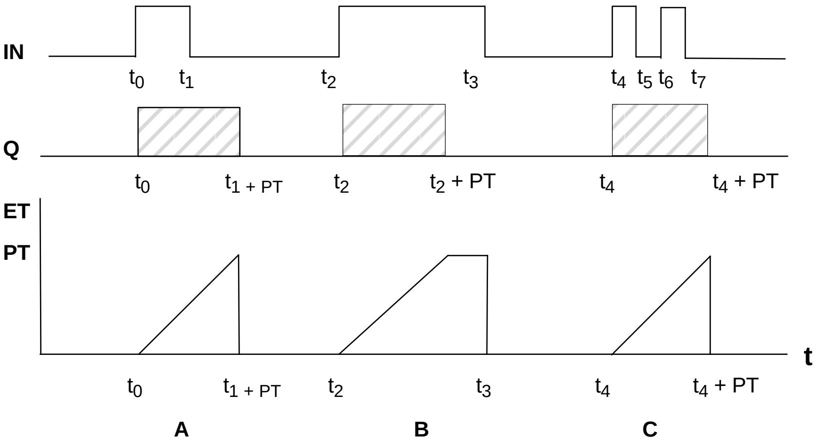
IN: **clock generator**
if a rising edge is detected at IN, a clock is generated having the period as defined in PT

PT: **clock period**
(16–bit value: 0 – 327.27s, 32–bit value: 0 –21,474,836.47s; resolution 10ms each) a clock having the period PT is caused for each rising edge at IN. A new rising edge detected at PT within the pulse period does not cause a new clock (see time chart, section C)

Q: **signal output**
is set for the period of PT as soon as a rising edge is detected at IN

ET: **elapsed time**
contains the elapsed period of the timer. If PT = ET, Q will be reset

Time Chart **TP**


IN
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7
Q
t_0 $t_{1 + PT}$ t_2 t_2 + PT t_4 t_4 + PT
ET
PT
t
t_0 $t_{1 + PT}$ t_2 t_3 t_4 t_4 + PT
A
B
C

A + B) Independent of the turn–on period of the IN signal, a clock is generated at the output Q having a length defined by PT. The function block TP is triggered if a rising edge is detected at the input IN.

C) A rising edge at the input IN does not have any influence during the processing of PT.

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is not available for FP1 or FP–M 0.9k.**

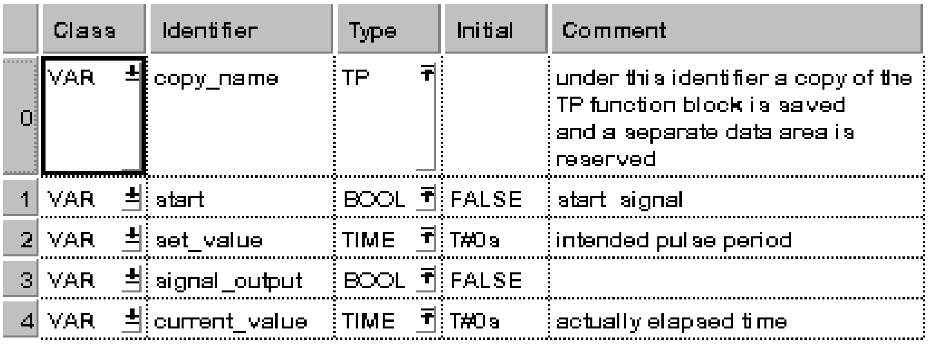
Data types

Data type	I/O	Function
BOOL	input IN	clock generated according to clock period at rising edge
TIME	input PT	clock period
BOOL	output Q	signal output
TIME	output ET	elapsed time

Example In this example the function block TP is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*. A separate data area is reserved for this copy.

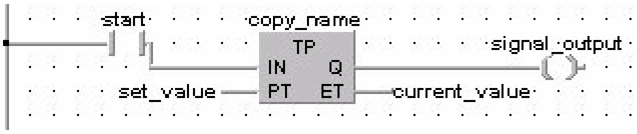


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	TP		under this identifier a copy of the TP function block is saved and a separate data area is reserved
1	VAR	start	BOOL	FALSE	start signal
2	VAR	set_value	TIME	T#0s	intended pulse period
3	VAR	signal_output	BOOL	FALSE	
4	VAR	current_value	TIME	T#0s	actually elapsed time

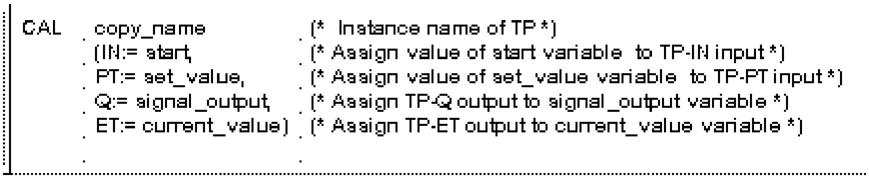
Body

If *start* is set (status = TRUE), the clock is emitted at *signal_output* until the *set_value* for the clock period is reached.

LD



IL



```
CAL    copy_name           (*  Instance name of TP *)
       (IN:= start,        (* Assign value of start variable  to TP-IN input *)
        PT:= set_value,    (* Assign value of set_value variable  to TP-PT input *)
        Q:= signal_output, (* Assign TP-Q output to signal_output variable *)
        ET:= current_value) (* Assign TP-ET output to current_value variable *)
```

The nomination *copy_name.IN* or *copy_name.ET* etc. has to be maintained in the IL.

(E_)TON Timer with switch–on delay

Description The function block TON allows you to program a switch–on delay. For TON declare the following:

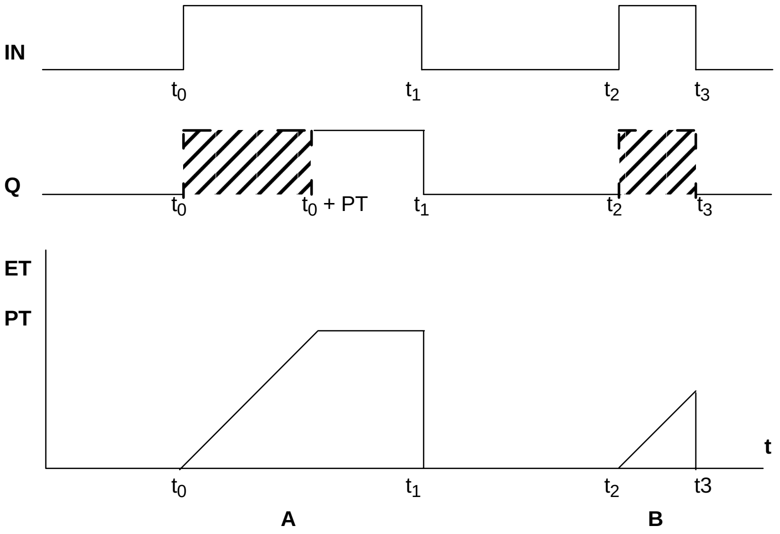
IN: **timer ON**
an internal timer is started for each rising edge detected at IN

PT: **switch on delay**
(16–bit value: 0 – 327.27s, 32–bit value: 0 – 21,474,836.47s; resolution 10ms each) the desired switch on delay is defined here(PT = preset time)

Q: **signal output**
is set if PT = ET

ET: **elapsed time**
indicates the current value of the elapsed time

Time Chart **TON**



A)Q is set delayed with the time defined in PT. Resetting is without any delay.

B)If the input IN is only set for the period of the delay time PT or even for a shorter period of time ($t_3 – t_2$ < PT), Q will not be set.

For the difference between the normal IEC function and the function with an enable input, page 24. You can find an example for the “function with enable” in the Online Help.

This function is not available for FP1 or FP–M 0.9k.

Data types

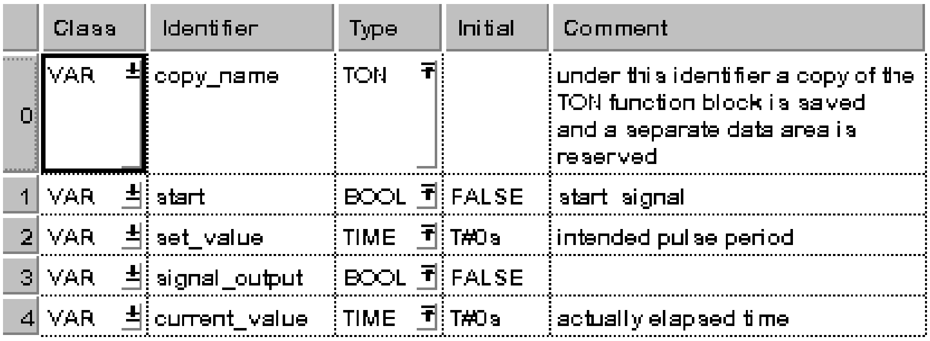
Data type	I/O	Function
BOOL (IN)	input	internal timer starts at rising edge
TIME (PT)	input	switch on delay
BOOL (Q)	output	signal output set if PT = ET
TIME (ET)	output	elapsed time

Example

In this example the function block TON is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*. A separate data area is reserved for this copy.

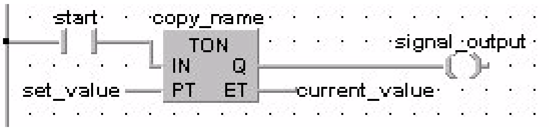


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	TON		under this identifier a copy of the TON function block is saved and a separate data area is reserved
1	VAR	start	BOOL	FALSE	start signal
2	VAR	set_value	TIME	T#0s	intended pulse period
3	VAR	signal_output	BOOL	FALSE	
4	VAR	current_value	TIME	T#0s	actually elapsed time

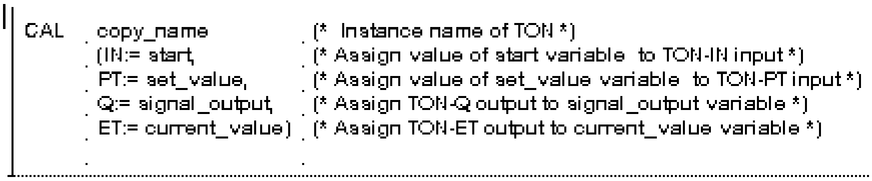
Body

If *start* is set (status = TRUE), the input signal is transferred to *signal_output* with a delay by the time period *set_value*.

LD



IL



```
CAL    copy_name            (*  Instance name of TON *)
       (IN:= start,         (* Assign value of start variable  to TON-IN input *)
        PT:= set_value,     (* Assign value of set_value variable  to TON-PT input *)
        Q:= signal_output,  (* Assign TON-Q output to signal_output variable *)
        ET:= current_value) (* Assign TON-ET output to current_value variable *)
```

The nomination *copy_name.IN* or *copy_name.ET* etc. has to be maintained in the IL.

(E_)TOF

Timer with switch–off delay

Description The function block TOF allows you to program a switch off delay, e.g. to switch off the ventilator of a machine at a later point of time than the machine itself. For TON declare the following:

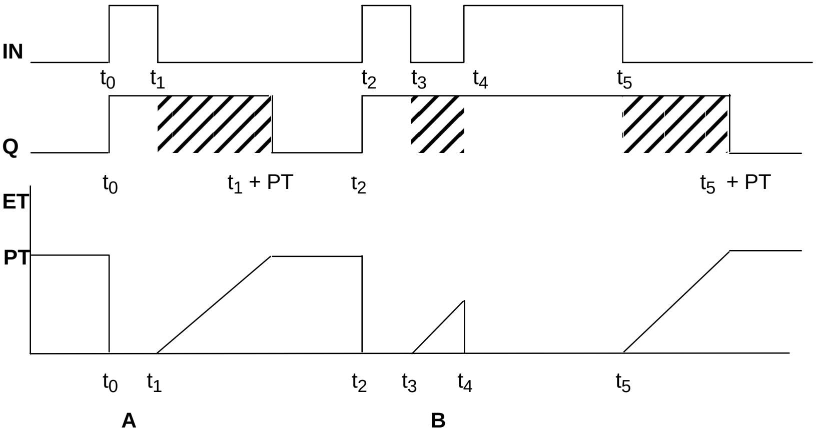
IN: **timer ON**
an internal time measuring device is started if a falling edge is detected at IN. If a rising edge is detected at IN before PT has reached its value, Q will not be switched off (see time chart, section B)

PT: **switch–off delay**
(16–bit value: 0 – 327.27s, 32–bit value: 0 – 21,474,836.47s; resolution 10ms each) the intended switch–off delay is defined here (PT = preset time)

Q: **signal output**
is reset if PT = ET

ET: **elapsed time**
represents the current value of the elapsed time

Time Chart **TOF**



A) Q is switched off with a delay corresponding to the time defined in PT. Switching on is carried out without delay.

B) If IN (as in the time chart on top for t_3 to t_4) is set prior to the lapse of the delay time PT, Q remains set (time chart for t_2 to t_3).

For the difference between the normal IEC function and the function with an enable input, see page 24. You can find an example for the “function with enable” in the Online Help.

☞ **This function is not available for FP1 or FP–M 0.9k.**

Data types

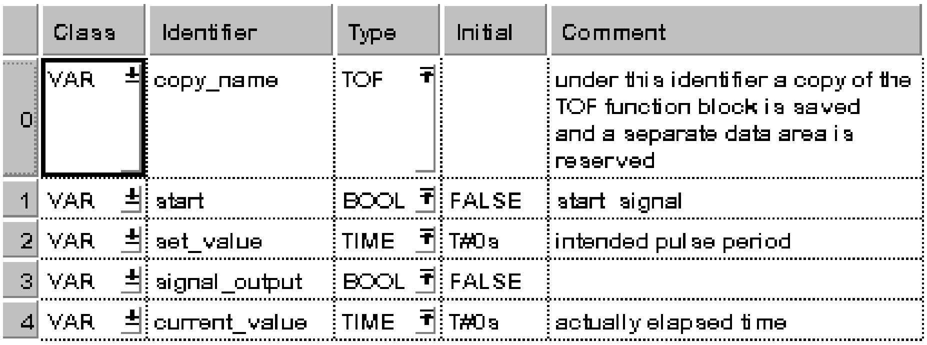
Data type	I/O	Function
BOOL (IN)	input	internal timer on at falling edge
TIME (PT)	input	switch off delay
BOOL (Q)	output	signal output reset if PT = ET
TIME (ET)	output	elapsed time

Example

In this example the function TOF is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*. A separate data area is reserved for this copy.

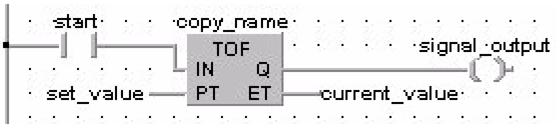


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name	TOF		under this identifier a copy of the TOF function block is saved and a separate data area is reserved
1	VAR	start	BOOL	FALSE	start signal
2	VAR	set_value	TIME	T#0s	intended pulse period
3	VAR	signal_output	BOOL	FALSE	
4	VAR	current_value	TIME	T#0s	actually elapsed time

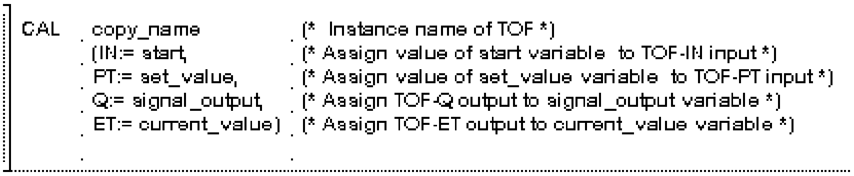
Body

If *start* is set, this signal is transferred to *signal_output* with a delay corresponding to the period of time *set_value*.

LD



IL



```
CAL   copy_name              (*  Instance name of TOF *)
      (IN:= start,           (* Assign value of start variable  to TOF-IN input *)
       PT:= set_value,       (* Assign value of set_value variable  to TOF-PT input *)
       Q:= signal_output,    (* Assign TOF-Q output to signal_output variable *)
       ET:= current_value)   (* Assign TOF-ET output to current_value variable *)
```

The nomination *copy_name.IN* or *copy_name.ET* etc. has to be maintained in the IL.

Part III

Matsushita Instructions

Matsushita Floating Point Instructions

The Matsushita floating point instructions are designed specifically for applications that require variables of the data type REAL. Most of these can be replaced by the more flexible IEC commands. By doing so you will reduce the number of commands with which you need to be familiar.

The following Matsushita floating point instructions are described in detail in this part because they are not easily duplicated with IEC instructions: F327_INT, F328_DINT, F333_FINT, F334_FRINT, F335_FSIGN, F337_RAD and F338_DEG.

For details and examples on the other Matsushita floating point instructions, see Online help. For quick reference, please refer to the table below.

Name	Function	Equivalent IEC function
F309_FMV	Constant floating point data move	E_MOVE
F310_FADD	Floating point data add	E_ADD
F311_FSUB	Floating point data subtract	E_SUB
F312_FMUL	Floating point data multiply	E_MUL
F313_FDIV	Floating point data divide	E_DIV
F314_FSIN	Floating point Sine operation	E_SIN
F315_FCOS	Floating point Cosine operation	E_COS
F316_FTAN	Floating point Tangent operation	E_TAN
F317_ASIN	Floating point Arcsine operation	E_ASIN
F318_ACOS	Floating point Arccosine operation	E_ACOS
F319_ATAN	Floating point Arctangent operation	E_ATAN
F320_LN	Floating point data natural logarithm	E_LN
F321_EXP	Floating point data exponent	E_EXP
F322_LOG	Floating point data logarithm	E_LOG
F323_PWR	Floating point data power	E_EXPT
F324_FSQR	Floating point data square root	E_SQRT
F325_FLT	16–bit integer → Floating point data	E_INT_TO_REAL
F326_DFLT	32–bit integer → Floating point data	E_DINT_TO_REAL
F329_FIX	Floating point data → 16–bit integer Rounding the first decimal point down	E_TRUNC_TO_INT
F330_DFIX	Floating point data → 32–bit integer Rounding the first decimal point down	E_TRUNC_TO_DINT
F331_ROFF	Floating point data → 16–bit integer Rounding the first decimal point off	E_REAL_TO_INT
F332_DROFF	Floating point data → 32–bit integer Rounding the first decimal point off	E_REAL_TO_DINT
F336_FABS	Floating point data absolute	E_ABS

Chapter 14

Counter, Timer Function Blocks

CT_FB Counter

Description Counters realized with the CT_FB function block are down counters. The count area **SV** (set value) is 1 to 32767. For the CT_FB function block declare the following:

Count: **count contact**
each time a rising edge is detected at **Count**, the value 1 is subtracted from the elapsed value **EV** until the value 0 is reached

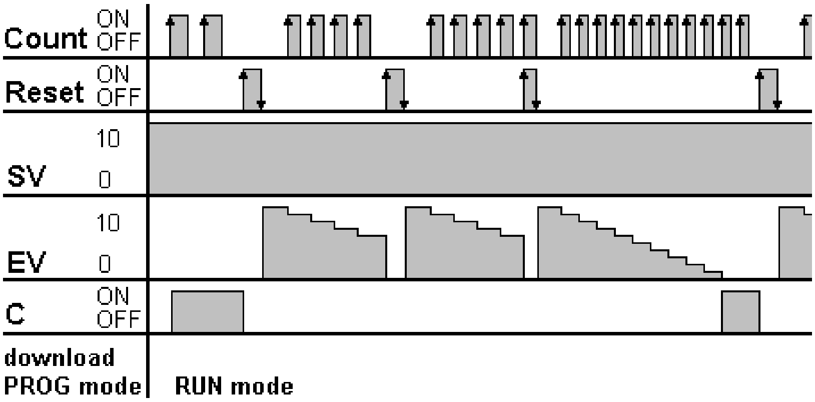
Reset: **reset contact**
each time a rising edge is detected at **Reset**, the value 0 is assigned to **EV** and the signal output **C** is reset; each time a falling edge is detected at **Reset**, the value at **SV** is assigned to **EV**

SV: **set value**
value of **EV** after a reset procedure

C: **signal output**
is set when **EV** becomes 0

EV: **elapsed value**
current counter value

Time Chart



- **In order to work correctly, the CT_FB function block needs to be reset each time before it is used.**
- **The number of available counters is limited and depends on the settings in the system registers 5 and 6. The compiler assigns a NUM* address to every counter instance. The addresses are assigned counting downwards, starting at the highest possible address.**
- **The Matsushita CT function (down counter) uses the same NUM* address area (Num* input). In order to avoid errors (address conflicts), the CT function and the CT_FB function block should not be used together in a project.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
CT_FB	x	x	x	x	x

x: available
–: not available

Data types

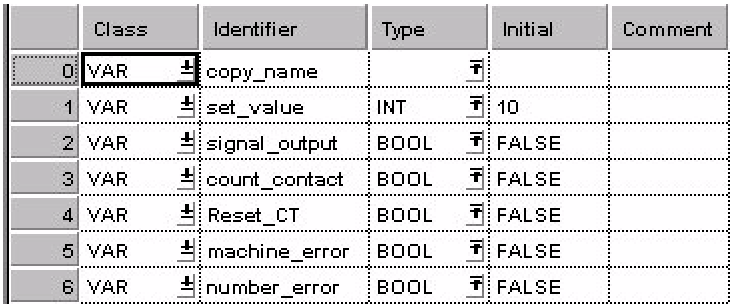
Variable	Data type	Function
Count	BOOL	count contact (down)
Reset	BOOL	reset contact
SV	INT, WORD	set value
C	BOOL	set when EV = 0
EV	INT, WORD	elapsed value

Example

In this example the function CT_FB is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *copy_name*, and a separate data area is reserved.

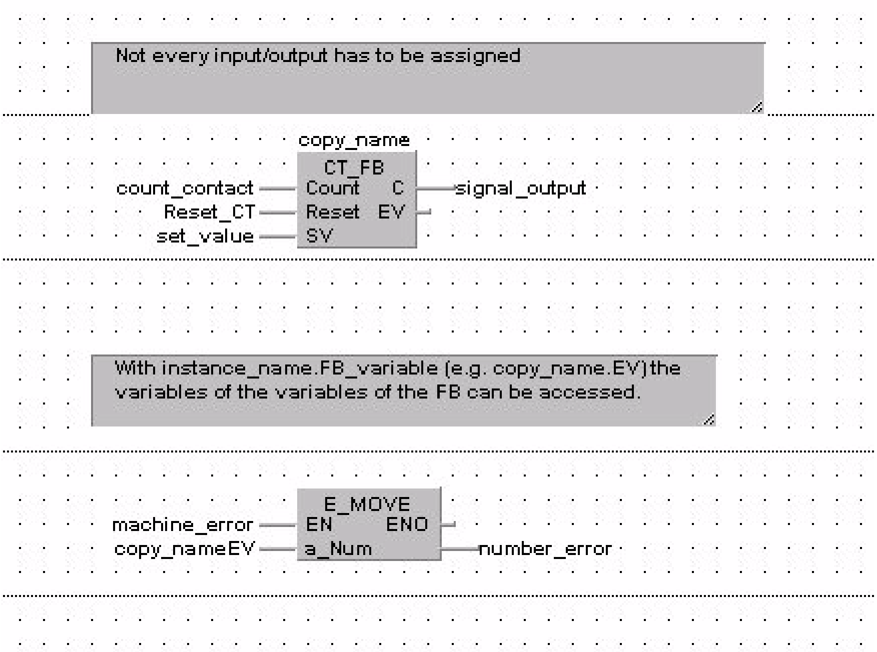


	Class	Identifier	Type	Initial	Comment
0	VAR	copy_name			
1	VAR	set_value	INT	10	
2	VAR	signal_output	BOOL	FALSE	
3	VAR	count_contact	BOOL	FALSE	
4	VAR	Reset_CT	BOOL	FALSE	
5	VAR	machine_error	BOOL	FALSE	
6	VAR	number_error	BOOL	FALSE	

Body

This example uses variables. You may also use constants for the input variables.

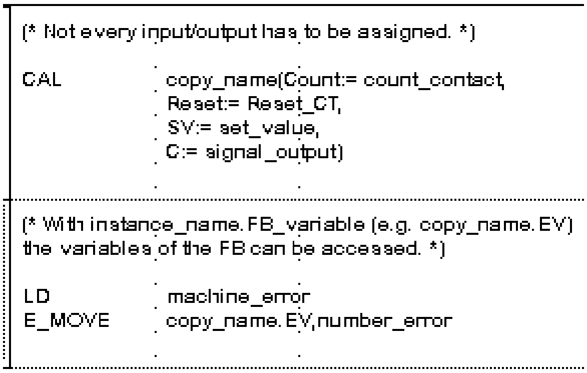
LD



Not every input/output has to be assigned

With instance_name.FB_variable (e.g. copy_name.EV)the variables of the variables of the FB can be accessed.

IL



```
(* Not every input/output has to be assigned. *)

CAL             copy_name(Count:= count_contact,
                Reset:= Reset_CT,
                SV:= set_value,
                C:= signal_output)

(* With instance_name.FB_variable (e.g. copy_name.EV)
the variables of the FB can be accessed. *)

LD              machine_error
E_MOVE          copy_name.EV,number_error
```

TM_1ms_FB Timer for 1ms intervals

Description This timer for 0.001s units works as an ON–delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON. For the TM_1ms_FB function block declare the following:

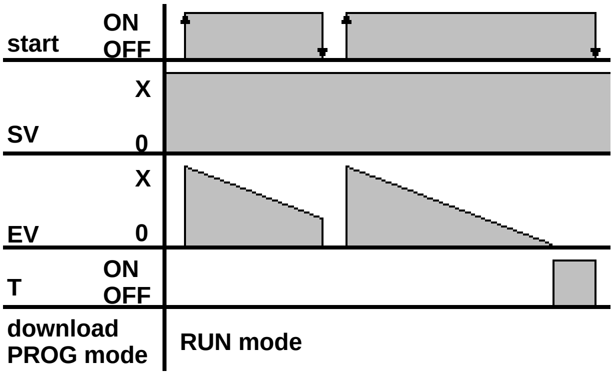
start: **start contact**
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV: **set value**
the defined ON–delay time (0 to 32.767s)

T: **timer contact**
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

EV: **elapsed value**
count value from which 1 is subtracted every 0.001s while the timer is running

Time Chart



☞

- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_1ms_FB	x	–	–	–	–

x: available
–: not available

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
start	x	x	x	x	x	x	–	–	–	–
T	–	x	x	x	–	–	–	–	–	–
SV, EV	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Data types

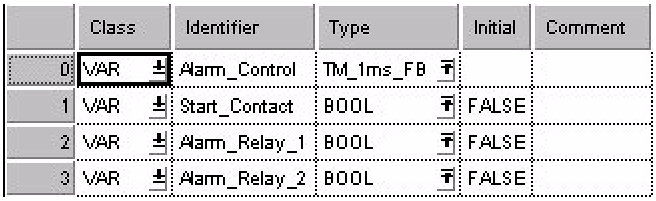
Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Example

In this example the functionTM_1ms_FB is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

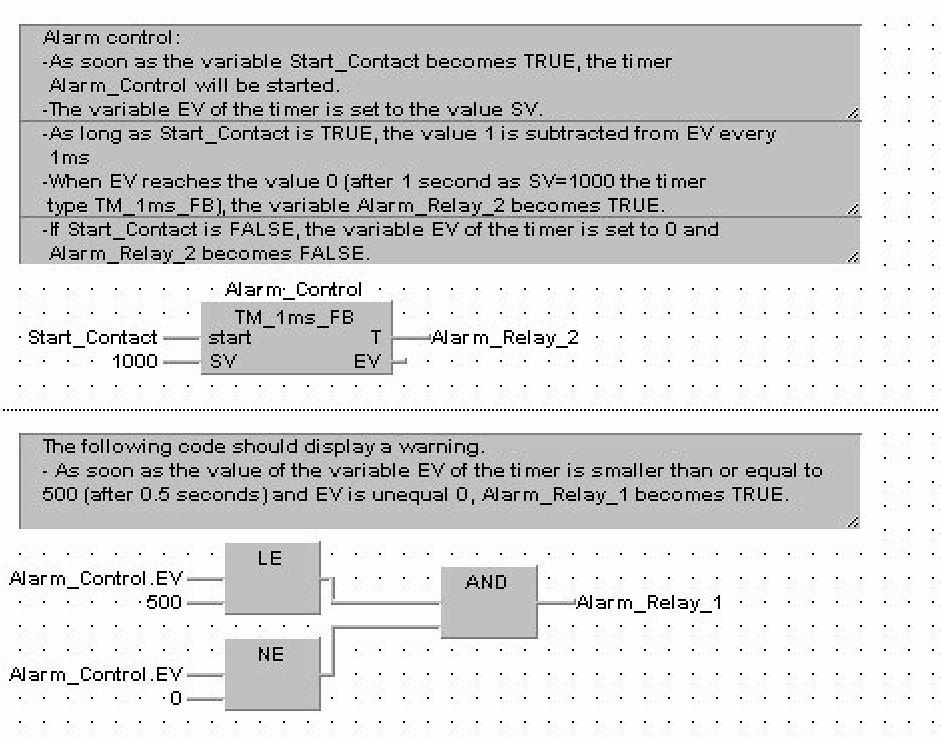
In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *Alarm_Control*, and a separate data area is reserved.



	Class	Identifier	Type	Initial	Comment
0	VAR	Alarm_Control	TM_1ms_FB		
1	VAR	Start_Contact	BOOL	FALSE	
2	VAR	Alarm_Relay_1	BOOL	FALSE	
3	VAR	Alarm_Relay_2	BOOL	FALSE	

Body This example uses variables. You may also use constants for the input variables.

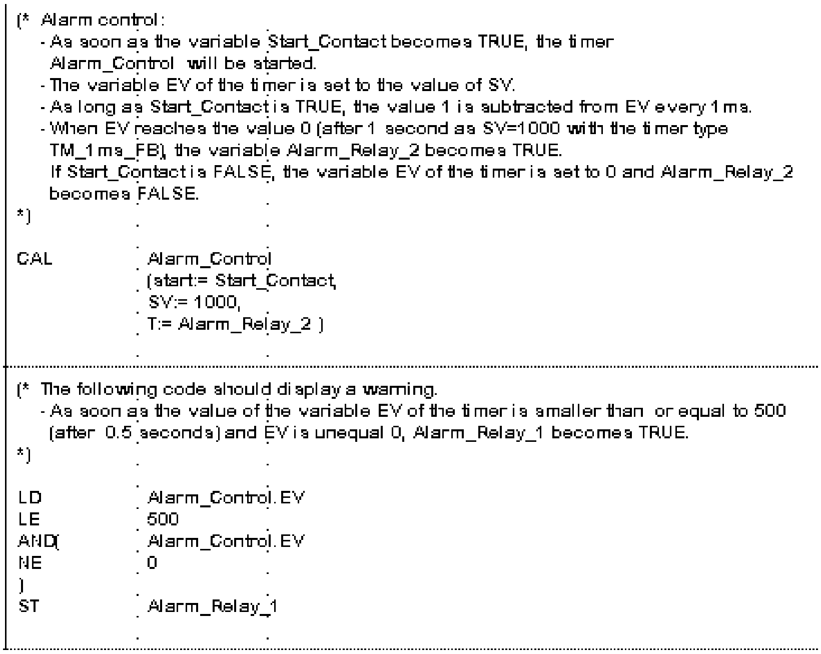
LD



Alarm control:
-As soon as the variable Start_Contact becomes TRUE, the timer
 Alarm_Control will be started.
-The variable EV of the timer is set to the value SV.
-As long as Start_Contact is TRUE, the value 1 is subtracted from EV every
 1ms
-When EV reaches the value 0 (after 1 second as SV=1000 the timer
 type TM_1ms_FB), the variable Alarm_Relay_2 becomes TRUE.
-If Start_Contact is FALSE, the variable EV of the timer is set to 0 and
 Alarm_Relay_2 becomes FALSE.

The following code should display a warning.
- As soon as the value of the variable EV of the timer is smaller than or equal to
500 (after 0.5 seconds) and EV is unequal 0, Alarm_Relay_1 becomes TRUE.

IL



```
(* Alarm control:
   - As soon as the variable Start_Contact becomes TRUE, the timer
     Alarm_Control  will be started.
   - The variable EV of the timer is set to the value of SV.
   - As long as Start_Contact is TRUE, the value 1 is subtracted from EV every 1 ms.
   - When EV reaches the value 0 (after 1  second as SV=1000 with the timer type
     TM_1ms_FB), the variable Alarm_Relay_2 becomes TRUE.
     If Start_Contact is FALSE, the variable EV of the timer is set to 0 and Alarm_Relay_2
     becomes FALSE.
*)

CAL          Alarm_Control
             (start:= Start_Contact,
             SV:= 1000,
             T:= Alarm_Relay_2 )
```

```
(* The following code should display a warning.
   - As soon as the value of the variable EV of the timer is smaller than  or equal to 500
     (after  0.5 seconds) and EV is unequal 0, Alarm_Relay_1 becomes TRUE.
*)

LD           Alarm_Control.EV
LE           500
AND(         Alarm_Control.EV
NE           0
)
ST           Alarm_Relay_1
```

TM_10ms_FB Timer for 10ms intervals

Description

This timer for 0.01s units works as an ON–delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON. For the TM_10ms_FB function block declare the following:

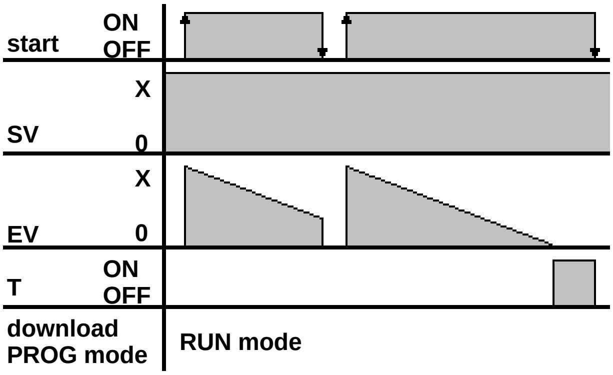
start: **start contact**
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV: **set value**
the defined ON–delay time (0 to 327.67s)

T: **timer contact**
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

EV: **elapsed value**
count value from which 1 is subtracted every 0.01s while the timer is running

Time Chart



- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_10ms_FB	x	x	x	x	x

x: available
–: not available

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
start	x	x	x	x	x	x	–	–	–	–
T	–	x	x	x	–	–	–	–	–	–
SV, EV	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Data types

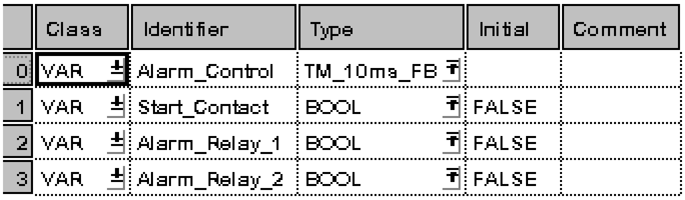
Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Example

In this example the function TM_10ms_FB is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

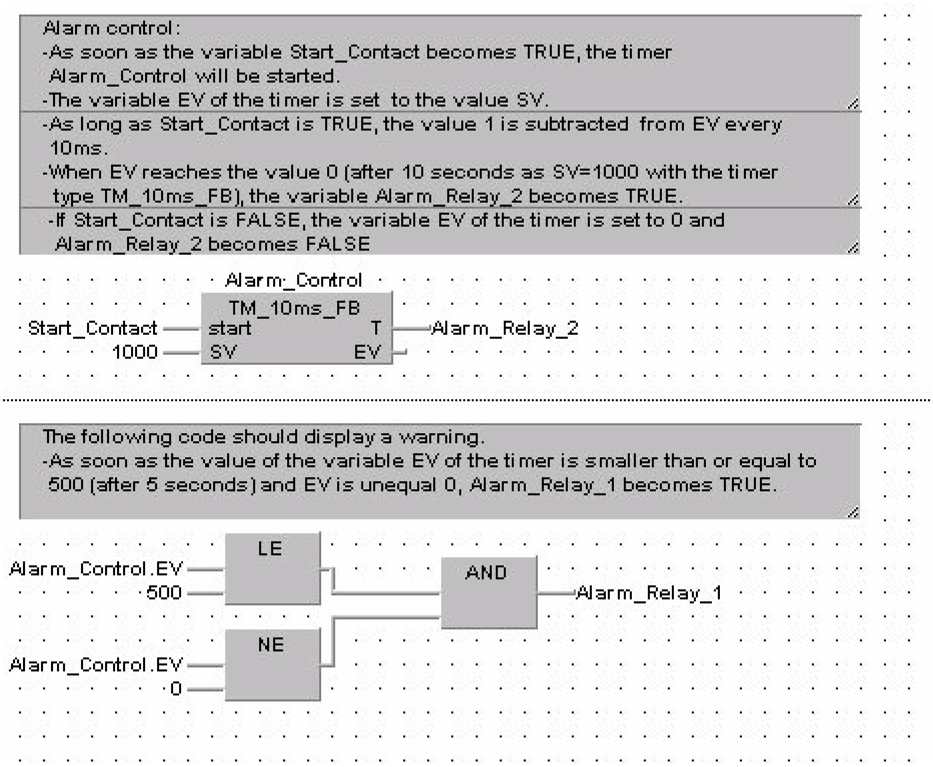
In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *Alarm_Control*, and a separate data area is reserved.



	Class	Identifier	Type	Initial	Comment
0	VAR	Alarm_Control	TM_10ms_FB		
1	VAR	Start_Contact	BOOL	FALSE	
2	VAR	Alarm_Relay_1	BOOL	FALSE	
3	VAR	Alarm_Relay_2	BOOL	FALSE	

Body This example uses variables. You may also use constants for the input variables.

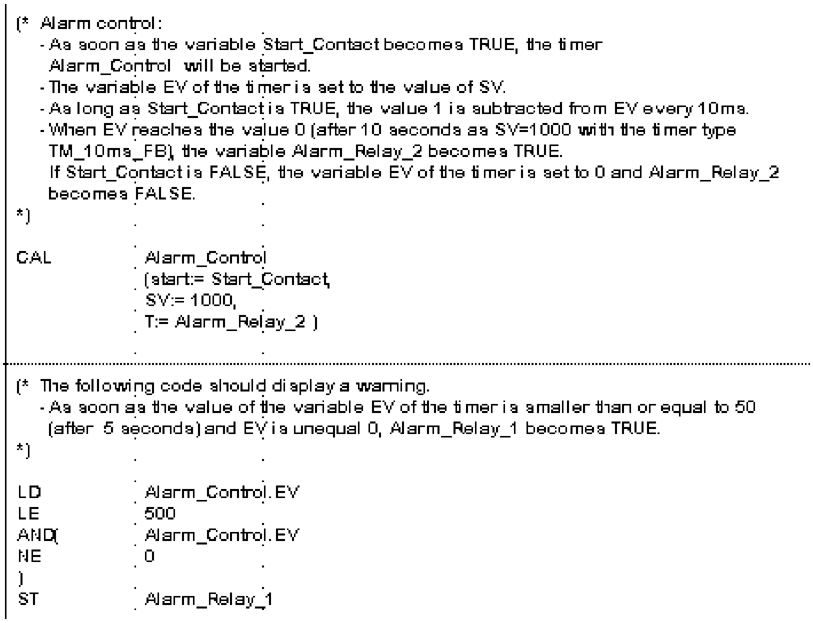
LD



Alarm control:
-As soon as the variable Start_Contact becomes TRUE, the timer Alarm_Control will be started.
-The variable EV of the timer is set to the value SV.
-As long as Start_Contact is TRUE, the value 1 is subtracted from EV every 10ms.
-When EV reaches the value 0 (after 10 seconds as SV=1000 with the timer type TM_10ms_FB), the variable Alarm_Relay_2 becomes TRUE.
-If Start_Contact is FALSE, the variable EV of the timer is set to 0 and Alarm_Relay_2 becomes FALSE

The following code should display a warning.
-As soon as the value of the variable EV of the timer is smaller than or equal to 500 (after 5 seconds) and EV is unequal 0, Alarm_Relay_1 becomes TRUE.

IL



```
(*  Alarm control:
    - As soon as the variable Start_Contact becomes TRUE, the timer
      Alarm_Control  will be started.
    - The variable EV of the timer is set to the value of SV.
    - As long as Start_Contact is TRUE, the value 1 is subtracted from EV every 10ms.
    - When EV reaches the value 0 (after 10 seconds as SV=1000 with the timer type
      TM_10ms_FB), the variable Alarm_Relay_2 becomes TRUE.
      If Start_Contact is FALSE, the variable EV of the timer is set to 0 and Alarm_Relay_2
      becomes FALSE.
*)

CAL             Alarm_Control
                (start:= Start_Contact,
                SV:= 1000,
                T:= Alarm_Relay_2 )

(*  The following code should display a warning.
    - As soon as the value of the variable EV of the timer is smaller than or equal to 50
      (after  5 seconds) and EV is unequal 0,  Alarm_Relay_1 becomes TRUE.
*)

LD              Alarm_Control.EV
LE              500
AND(            Alarm_Control.EV
NE              0
)
ST              Alarm_Relay_1
```

TM_100ms_FB Timer for 100ms intervals

Description This timer for 0.1s units works as an ON–delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON. For the TM_100ms_FB function block declare the following:

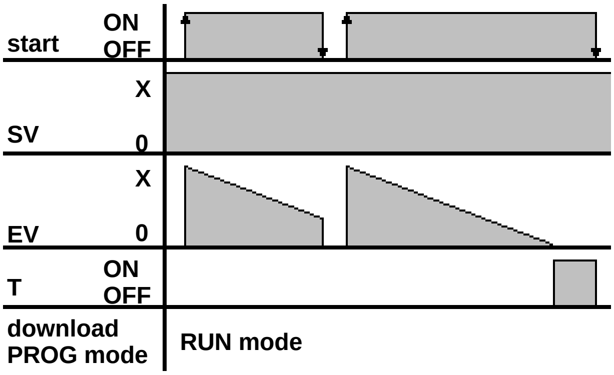
start: **start contact**
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV: **set value**
the defined ON–delay time (0 to 3276.7s)

T: **timer contact**
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

EV: **elapsed value**
count value from which 1 is subtracted every 0.1s while the timer is running

Time Chart



☞

- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_100ms_FB	x	x	x	x	x

x: available
–: not available

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
start	x	x	x	x	x	x	–	–	–	–
T	–	x	x	x	–	–	–	–	–	–
SV, EV	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Data types

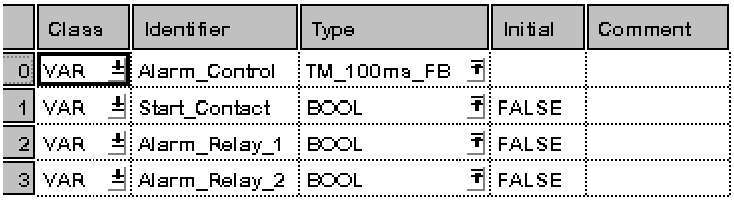
Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Example

In this example the functionTM_100ms_FB is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

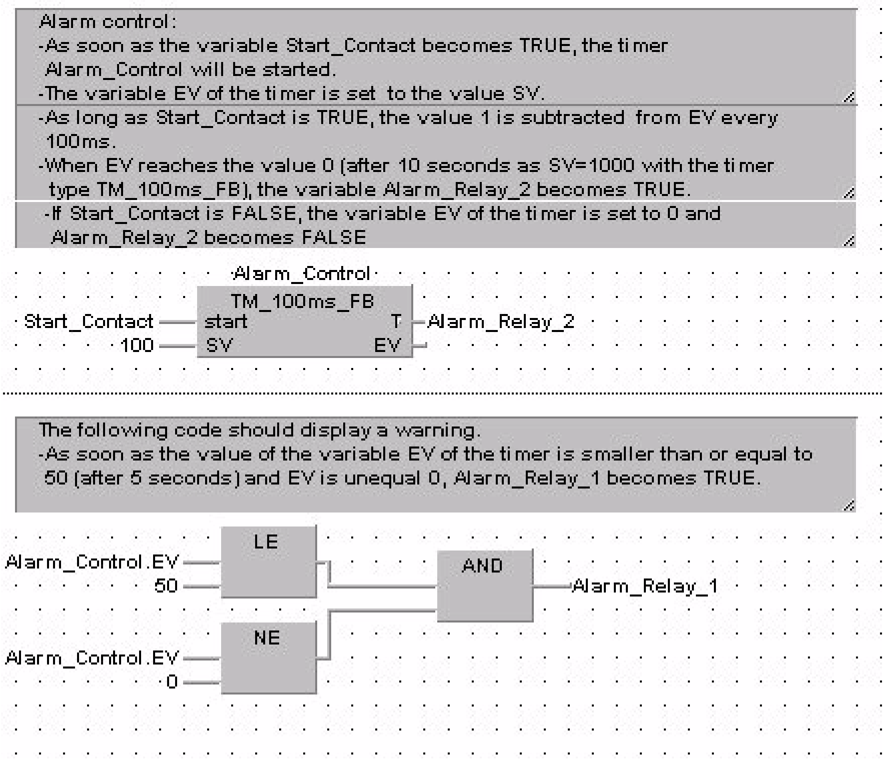
In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *Alarm_Control*, and a separate data area is reserved.



	Class	Identifier	Type	Initial	Comment
0	VAR	Alarm_Control	TM_100ms_FB		
1	VAR	Start_Contact	BOOL	FALSE	
2	VAR	Alarm_Relay_1	BOOL	FALSE	
3	VAR	Alarm_Relay_2	BOOL	FALSE	

Body This example uses variables. You may also use constants for the input variables.

LD



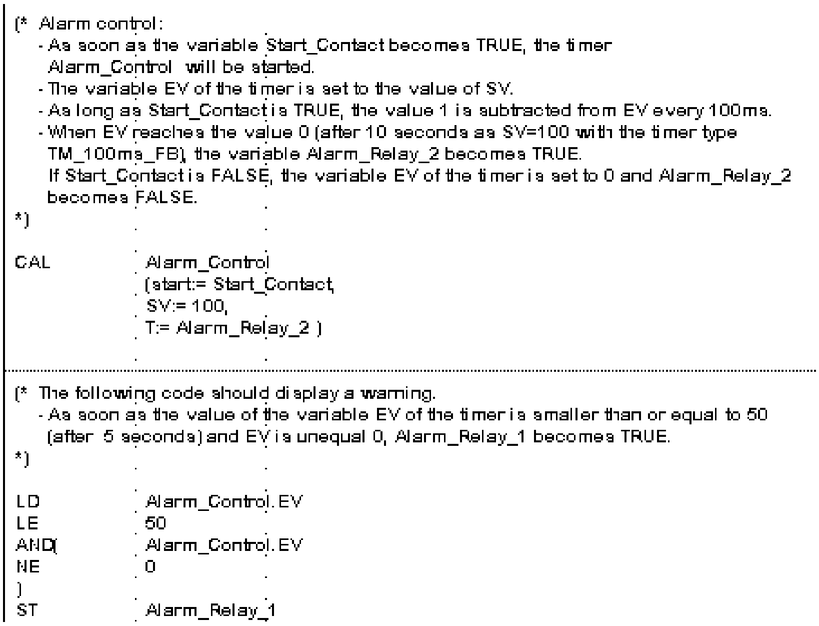
Alarm control:
-As soon as the variable Start_Contact becomes TRUE, the timer Alarm_Control will be started.
-The variable EV of the timer is set to the value SV.

-As long as Start_Contact is TRUE, the value 1 is subtracted from EV every 100ms.
-When EV reaches the value 0 (after 10 seconds as SV=1000 with the timer type TM_100ms_FB), the variable Alarm_Relay_2 becomes TRUE.

-If Start_Contact is FALSE, the variable EV of the timer is set to 0 and Alarm_Relay_2 becomes FALSE

The following code should display a warning.
-As soon as the value of the variable EV of the timer is smaller than or equal to 50 (after 5 seconds) and EV is unequal 0, Alarm_Relay_1 becomes TRUE.

IL



```
(*  Alarm control:
    - As soon as the variable Start_Contact becomes TRUE, the timer
      Alarm_Control  will be started.
    - The variable EV of the timer is set to the value of SV.
    - As long as Start_Contact is TRUE, the value 1 is subtracted from EV every 100ms.
    - When EV reaches the value 0 (after 10 seconds as SV=100 with the timer type
      TM_100ms_FB), the variable Alarm_Relay_2 becomes TRUE.
      If Start_Contact is FALSE, the variable EV of the timer is set to 0 and Alarm_Relay_2
      becomes FALSE.
*)

CAL          Alarm_Control
             (start:= Start_Contact,
             SV:= 100,
             T:= Alarm_Relay_2 )

(*  The following code should display a warning.
    - As soon as the value of the variable EV of the timer is smaller than or equal to 50
      (after 5 seconds) and EV is unequal 0, Alarm_Relay_1 becomes TRUE.
*)

LD           Alarm_Control.EV
LE           50
AND(         Alarm_Control.EV
NE           0
)
ST           Alarm_Relay_1
```

TM_1s_FB Timer for 1s intervals

Description

This timer for 1s units works as an ON–delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON. For the TM_1s_FB function block declare the following:

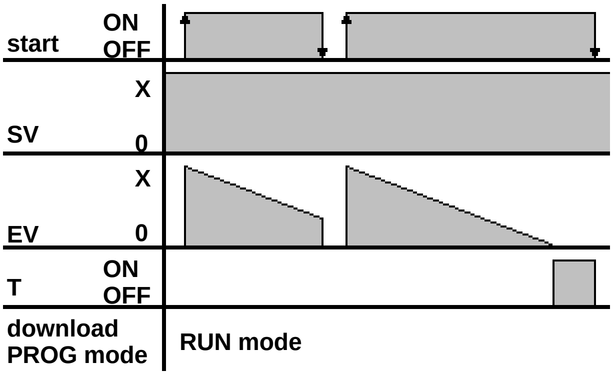
start: **start contact**
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

SV: **set value**
the defined ON–delay time (0 to 32767s)

T: **timer contact**
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

EV: **elapsed value**
count value from which 1 is subtracted every 1s while the timer is running

Time Chart



☞

- **The number of available timers is limited and depends on the settings in the system registers 5 and 6.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_1s_FB	x	x	x	x	x

x: available
–: not available

Data types

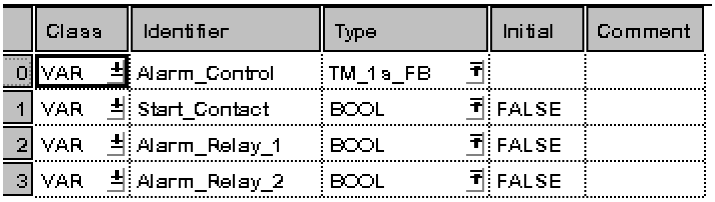
Variable	Data type	Function
start	BOOL	start contact
SV	INT, WORD	set value
T	BOOL	timer contact
EV	INT, WORD	elapsed value

Example

In this example the function TM_1s_FB is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.
This also includes the function block (FB) itself. By declaring the FB you create a copy of the original FB. This copy is saved under *Alarm_Control*, and a separate data area is reserved.

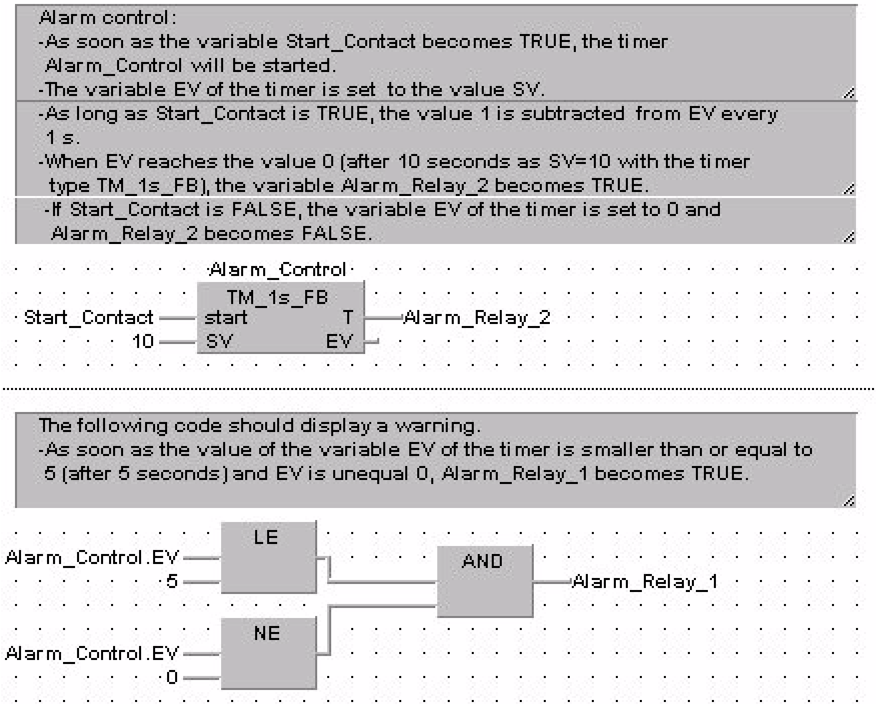


	Class	Identifier	Type	Initial	Comment
0	VAR	Alarm_Control	TM_1s_FB		
1	VAR	Start_Contact	BOOL	FALSE	
2	VAR	Alarm_Relay_1	BOOL	FALSE	
3	VAR	Alarm_Relay_2	BOOL	FALSE	

Body

This example uses variables. You may also use constants for the input variables.

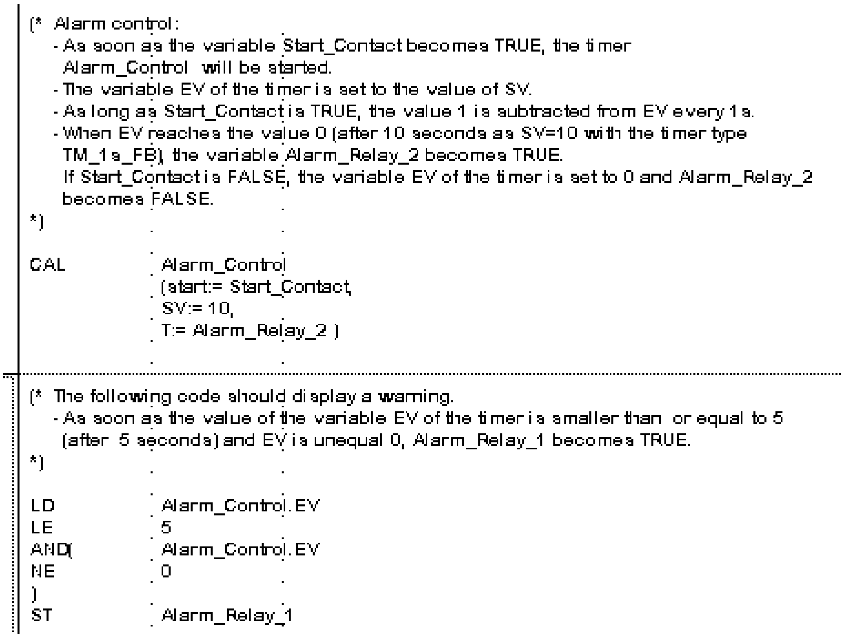
LD



Alarm control:
-As soon as the variable Start_Contact becomes TRUE, the timer Alarm_Control will be started.
-The variable EV of the timer is set to the value SV.
-As long as Start_Contact is TRUE, the value 1 is subtracted from EV every 1 s.
-When EV reaches the value 0 (after 10 seconds as SV=10 with the timer type TM_1s_FB), the variable Alarm_Relay_2 becomes TRUE.
-If Start_Contact is FALSE, the variable EV of the timer is set to 0 and Alarm_Relay_2 becomes FALSE.

The following code should display a warning.
-As soon as the value of the variable EV of the timer is smaller than or equal to 5 (after 5 seconds) and EV is unequal 0, Alarm_Relay_1 becomes TRUE.

IL



```
(* Alarm control:
   - As soon as the variable Start_Contact becomes TRUE, the timer
     Alarm_Control  will be started.
   - The variable EV of the timer is set to the value of SV.
   - As long as Start_Contact is TRUE, the value 1 is subtracted from EV every 1 s.
   - When EV reaches the value 0 (after 10 seconds as SV=10 with the timer type
     TM_1s_FB), the variable Alarm_Relay_2 becomes TRUE.
     If Start_Contact is FALSE, the variable EV of the timer is set to 0 and Alarm_Relay_2
     becomes FALSE.
*)

CAL          Alarm_Control
             (start:= Start_Contact,
             SV:= 10,
             T:= Alarm_Relay_2 )

(* The following code should display a warning.
   - As soon as the value of the variable EV of the timer is smaller than  or equal to 5
     (after  5 seconds) and EV is unequal 0, Alarm_Relay_1 becomes TRUE.
*)

LD           Alarm_Control.EV
LE           5
AND(         Alarm_Control.EV
NE           0
)
ST           Alarm_Relay_1
```

Chapter 15

Data Transfer Instructions

F0_MV

16–bit data move

Steps	5

Description The 16–bit data or 16–bit equivalent constant specified by **s** is copied to the 16–bit area specified by **d**, if the trigger **EN** is in the ON–state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F0	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	source 16–bit area
d	INT, WORD	destination 16–bit area

The variables **s** and **d** have to be of the same data type.

Operands

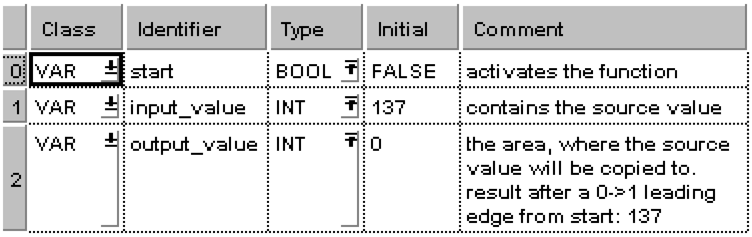
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

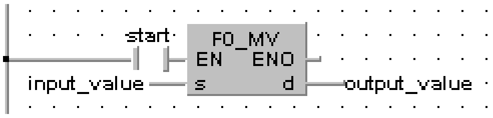


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	INT	137	contains the source value
2	VAR	output_value	INT	0	the area, where the source value will be copied to. result after a 0->1 leading edge from start: 137

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F0_MV(input_value, output_value);
END_IF;
```

F1_DMV

32–bit data move

Steps 7

Description The 32–bit data or 32–bit equivalent constant specified by **s** is copied to the 32–bit area specified by **d**, if the trigger **EN** is in the ON–state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F1	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DINT, DWORD	source 32–bit area
d	DINT, DWORD	destination 32–bit area

The variables **s** and **d** have to be of the same data type.

Operands

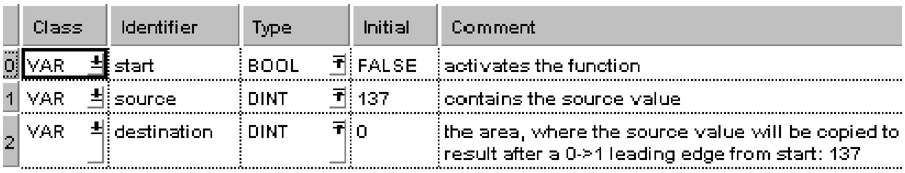
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

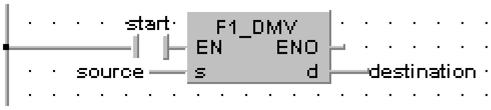


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source	DINT	137	contains the source value
2	VAR	destination	DINT	0	the area, where the source value will be copied to result after a 0->1 leading edge from start: 137

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F1_DMV(source, destination);
END_IF;
```

F2_MVN

16–bit data inversions and move

Steps	5

Description The 16–bit data or 16–bit equivalent constant specified by **s** is inverted and transferred to the 16–bit area specified by **d** if the trigger **EN** is in the ON–state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F2	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	source 16–bit area to be inverted
d	INT, WORD	destination 16–bit area

The variables **s** and **d** have to be of the same data type.

Operands

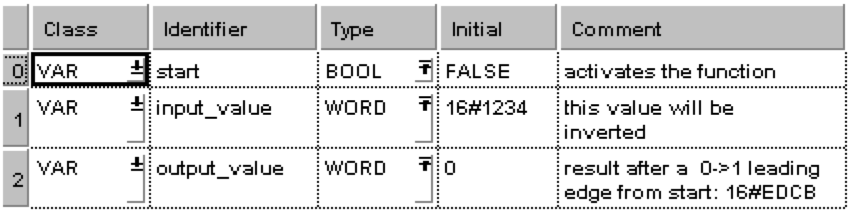
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

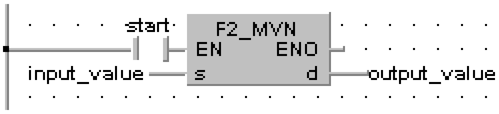


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	16#1234	this value will be inverted
2	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#EDCB

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F2_MVN(input_value, output_value);
END_IF;
```

F3_DMVN

32–bit data inversions and move

Steps	7

Description The 32–bit data or 32–bit equivalent constant specified by **s** is inverted and transferred to the 32–bit area specified by **d** if the trigger **EN** is in the ON–state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F3	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DINT, DWORD	source 32–bit area to be inverted
d	DINT, DWORD	destination 32–bit area

The variables **s** and **d** have to be of the same data type.

Operands

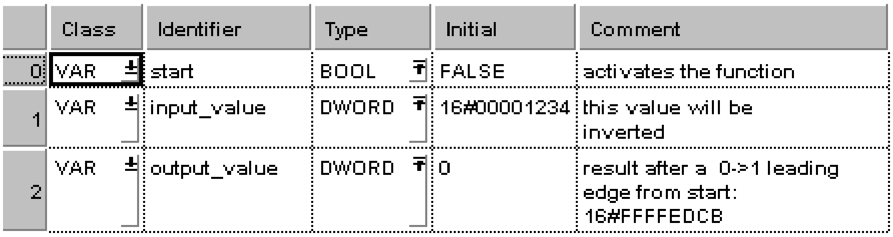
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

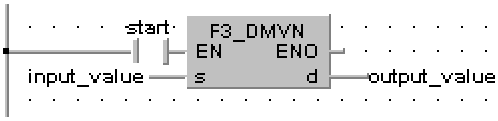


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	DWORD	16#00001234	this value will be inverted
2	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#FFFFEDCB

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F3_DMVN(input_value, output_value);
END_IF;
```

F5_BTM

Bit data move

Steps	7

Description

1 bit of the 16–bit data or constant value specified by **s** is copied to a bit of the 16–bit area specified by **d** according to the content specified by **n** if the trigger **EN** is in the ON–state. When the 16–bit equivalent constant is specified by **s**, the bit data move operation is performed internally converting it to 16–bit binary expression. The operand **n** specifies the bit number as follows:

- Bit No. 0 to 3: source bit No. (16#0 to 16#F)
- Bit No. 8 to 11: destination bit No. (16#0 to 16#F)
- The bits 4 to 7 are fixed to move one bit and 12 to 15 are invalid

For example, reading from the right, **n** = 16#C01 would move from bit position one, one bit to bit position 12 (C).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F5	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	source 16–bit area
n	INT, WORD	specifies source and destination bit positions
d	INT, WORD	destination 16–bit area

The variables **s** and **d** have to be of the same data type.

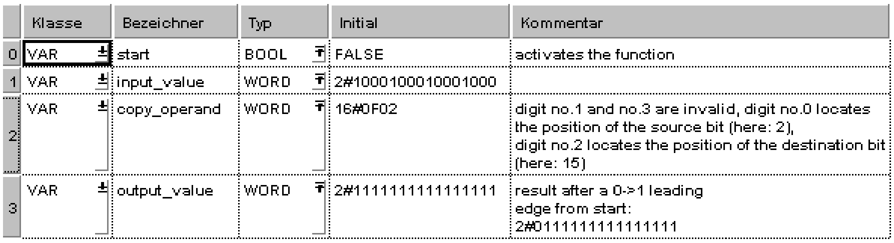
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s, n	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

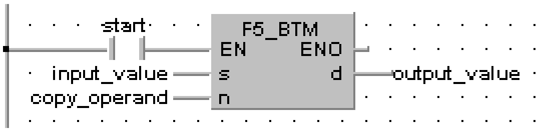
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Klasse	Bezeichner	Typ	Initial	Kommentar
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#1000100010001000	
2	VAR	copy_operand	WORD	16#0F02	digit no.1 and no.3 are invalid, digit no.0 locates the position of the source bit (here: 2), digit no.2 locates the position of the destination bit (here: 15)
3	VAR	output_value	WORD	2#1111111111111111	result after a 0->1 leading edge from start: 2#0111111111111111

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F5_BTM( s:= input_value,
             n:= copy_operand,
             d=> output_value);
END_IF;
```

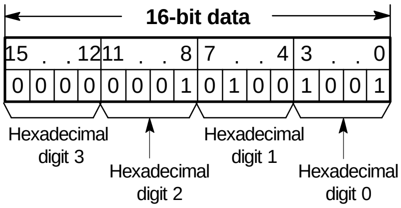
F6_DGT

Digit data move

Steps 7

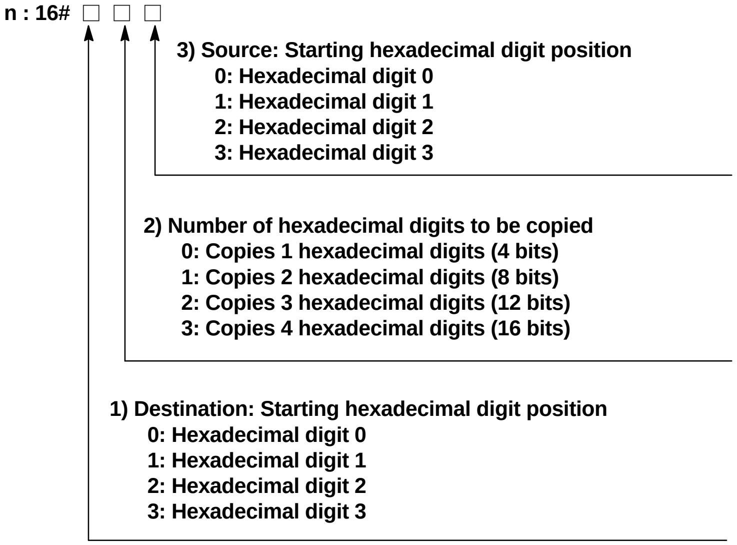
Description The hexadecimal digits in the 16-bit data or in the 16-bit equivalent constant specified by **s** are copied to the 16-bit area specified by **d** as specified by **n**.

Digits are units of 4 bits used when handling data. With this instruction, 16–bit data is separated into four digits. The digits are called in order hexadecimal digit 0, digit 1, digit 2 and digit 3, beginning from the least significant four bits:



15 . . 12				11 . . 8				7 . . 4				3 . . 0			
0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1

n specifies the 3) source hexadecimal digit position, the 2) number of digits and the 1) destination hexadecimal digit position to be copied using hexadecimal data as follows:



n : 16# □ □ □

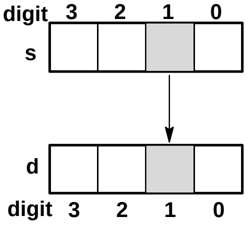
3) Source: Starting hexadecimal digit position
0: Hexadecimal digit 0
1: Hexadecimal digit 1
2: Hexadecimal digit 2
3: Hexadecimal digit 3

2) Number of hexadecimal digits to be copied
0: Copies 1 hexadecimal digits (4 bits)
1: Copies 2 hexadecimal digits (8 bits)
2: Copies 3 hexadecimal digits (12 bits)
3: Copies 4 hexadecimal digits (16 bits)

1) Destination: Starting hexadecimal digit position
0: Hexadecimal digit 0
1: Hexadecimal digit 1
2: Hexadecimal digit 2
3: Hexadecimal digit 3

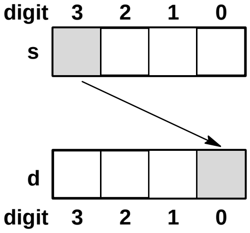
Following are some patterns of digit transfer based on the specification of **n**.

- When hexadecimal digit 1 of the source is copied to hexadecimal digit 1 of the destination:



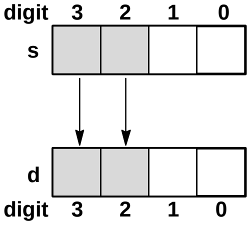
Specify n: 16# 1 0 1

- When hexadecimal digit 3 of the source is copied to hexadecimal digit 0 of the destination:



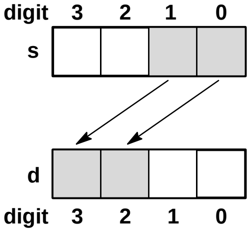
Specify n: 16# 0 0 3 (Short form: 16#3)

- When multiple hexadecimal digits (hexadecimal digits 2 and 3) of the source are copied to multiple hexadecimal digits (hexadecimal digits 2 and 3) of the destination:



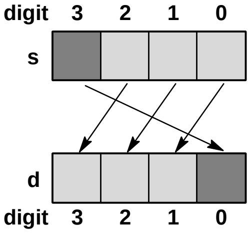
Specify n: 16# 2 1 2

- When multiple hexadecimal digits (hexadecimal digits 0 and 1) of the source are copied to multiple hexadecimal digits (hexadecimal digits 2 and 3) of the destination:



Specify n: 16# 2 1 0

- When 4 hexadecimal digits (hexadecimal digits 0 to 3) of the source are copied to 4 hexadecimal digits (hexadecimal digits 0 to 3) of the destination:



Specify n: 16# 1 3 0

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F6	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	16–bit area source
n	INT, WORD	Specifies source and destination hexadecimal digit position and number of hexadecimal digits
d	INT, WORD	16–bit area destination

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s, n	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

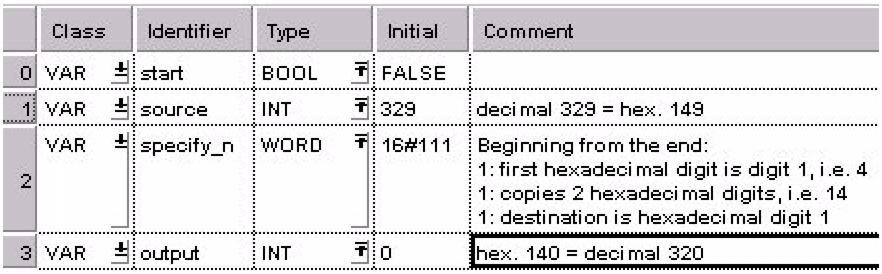
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

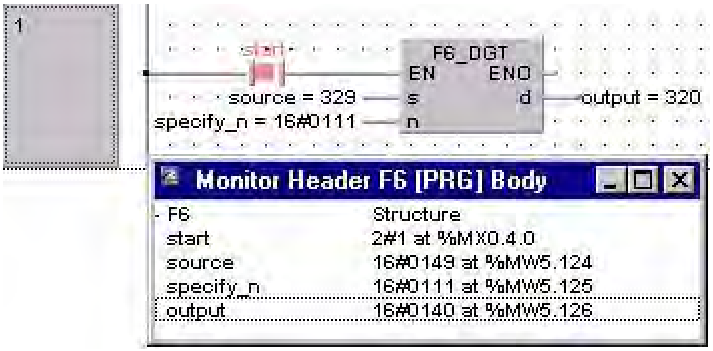


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	
1	VAR	source	INT	329	decimal 329 = hex. 149
2	VAR	specify_n	WORD	16#111	Beginning from the end: 1: first hexadecimal digit is digit 1, i.e. 4 1: copies 2 hexadecimal digits, i.e. 14 1: destination is hexadecimal digit 1
3	VAR	output	INT	0	hex. 140 = decimal 320

Body

When the variable *start* is set to TRUE, the function is executed. The values for *source* and *output* in the Monitor Header of the ladder diagram body have been set to display the hexadecimal value by activating the Hex button in the tool bar.

LD



Monitor Header F6 [PRG] Body

F6	Structure
start	2#1 at %MX0.4.0
source	16#0149 at %MW5.124
specify_n	16#0111 at %MW5.125
output	16#0140 at %MW5.126

ST

```
IF start THEN
      F6_DGT( s:= source,
             n:= specify_n,
             d=> output);
END_IF;
```

F10_BKMV

Block transfer

Steps 7

Description The data block specified by the 16–bit starting area specified by **s1** and the 16–bit ending area specified by **s2** are copied to the block starting from the 16–bit area specified by **d** if the trigger **EN** is in the ON–state. The operands **s1** and **s2** should be:

- in the same operand
- **s1** ≤ **s2**

Whenever **s1**, **s2** and **d** are in the same data area:

- **s1** = **d**: data will be recopied to the same data area.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F10	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	starting 16–bit area, source
s2	INT, WORD	ending 16–bit area, source
d	INT, WORD	starting 16–bit area, destination

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	–
d	–	x	x	x	x	x	x	x	x	–

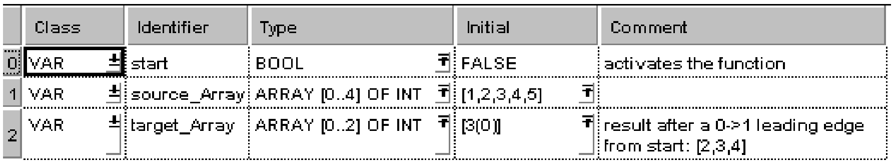
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

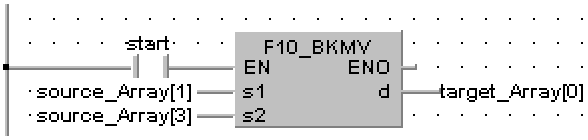


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_Array	ARRAY [0..4] OF INT	[1,2,3,4,5]	
2	VAR	target_Array	ARRAY [0..2] OF INT	[3(0)]	result after a 0->1 leading edge from start: [2,3,4]

Body

When the variable *start* changes from FALSE to TRUE, the function is carried out. It moves the data block starting at the 16–bit area specified by s1 and ending at the 16–bit area specified by s2 to the 16–bit area specified by s3.

LD



ST

```
IF start THEN
      F10_BKMV( s1_Start:= source_Array[1],
             s2_End:= source_Array[3],
             d_Start=> target_Array[0]);
END_IF;
```

F11_COPY

Block copy

Steps 7

Description

The 16–bit equivalent constant or 16–bit area specified by **s** is copied to all 16–bit areas of the block specified by **d1** and **d2** if the trigger **EN** is in the ON–state. The operands **d1** and **d2** should be:

- in the same operand
- d1 ≤ d2

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F11	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	source 16–bit area
d1	INT, WORD	starting 16–bit area, destination
d2	INT, WORD	ending 16–bit area, destination

The variables **s, d1** and **d2** have to be of the same data type.

Operands

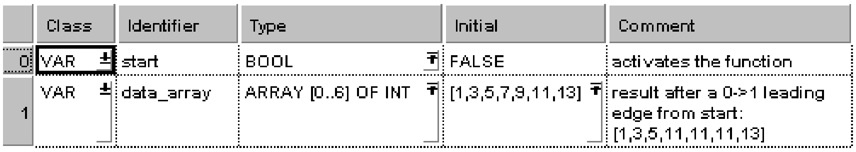
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d1, d2	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

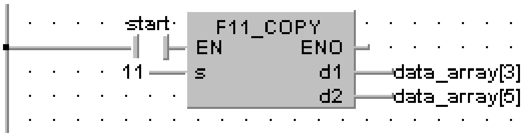


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data_array	ARRAY [0..6] OF INT	[1,3,5,7,9,11,13]	result after a 0->1 leading edge from start: [1,3,5,11,11,11,13]

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      (* Copy the value 11 to data_array[3], *)
      (* data_array[4] and data_array[5] *)
      F11_COPY( s:= 11,
             d1_Start=> data_array[3],
             d2_End=> data_array[5]);
END_IF;
```

F12_EPRD

EEPROM read from memory

Steps **11**

Description This instruction is used to read information from the EEPROM. Before executing the F12_EPRD instruction, make sure that you have valid data in the EEPROM memory location being read to the destination area. Otherwise the values being read will not make any sense. Also ensure that there are at least 64 free data registers (1 block = 64 words (DTs)) reserved for the destination area.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F12	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
EN	BOOL	Activation of the function block (when EN has the state TRUE, the function block will be executed at every PLC scan)
s1	DINT,D WORD	EEPROM start block number
s2	DINT, DWORD	Number of blocks to write (1 block = 64 words (DTs))
d	INT, WORD	DT start address for information to be written
ENO	BOOL	When the function block was executed, ENO is set to TRUE. Helpful at cascading of function blocks with EN–functionality

☞ **One of the two inputs ’s1’ or ’s2’ has to be assigned a constant number value.**

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2	x	x	x	–	x	x	x	–	–	x
d	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	–	–	–	–	–	x	–	–	–

x: available
–: not available

PLC–specific information

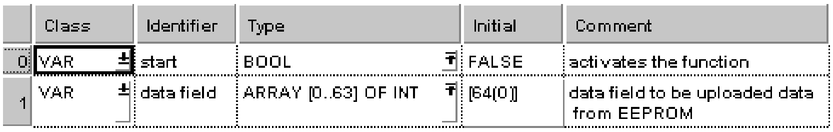
PLC type	FP0 2,7k C10/C14/C16	FP0 5k C32	FP0 10k T32CP
Block size (1 block)	64 words (64 x 16 bit)	64 words (64 x 16 bit)	64 words (64 x 16 bit)
EEPROM start block number	0 to 9	0 to 95	0 to 255
Number of blocks to be read / written each execution	1 to 2	1 to 8	1 to 255
Write duration (Additional scan time)	20 ms each block	5 ms each block	5 ms each block
Read duration (Additional scan time)	Less than 1 ms each block	Less than 1ms each block	Less than 1ms each block
Max number of writing events	100,000	10,000	10,000
Note Power down RUN –> Prog mode changes are also counted			
Max read times	No limit	No limit	No limit

Example

In this example the function F12_EPRD is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

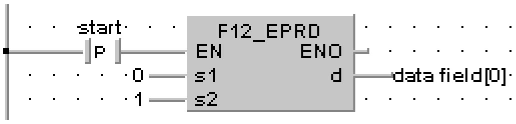


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data field	ARRAY [0..63] OF INT	[64(0)]	data field to be uploaded data from EEPROM

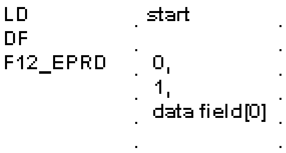
Body

When the variable *start* changes from FALSE to TRUE, the function is carried out. The function reads the first block (= 64 words) after start block number 0 from the EEPROM and writes the information into the data fields from *data-_field[0]* until *data-_field[63]*.

LD



IL



```
LD          start
DF
F12_EPRD     0,
             1,
             data field[0]
```

P13_EPWT

EEPROM write to memory

Steps **11**

Description

These instructions are used to save your PID profiles, timer profiles, counter profiles or positioning profiles ... into the built–in EEPROM. The EEPROM memory is not the same as the hold area. The hold area stores data in real time. Whenever the power shuts down, the hold data is stored in the EEPROM memory.

The P13_EPWT instruction sends data into the EEPROM only when the instruction is executed. It also has a limitation of the number of times you can write to it (see table on PLC–specific information). You must make sure that the P13_EPWT instruction will not be executed more often than the specified number of writes.

For example, if you execute P13_EPWT with R901A relay (pulse time 0.1s), the EEPROM will become inoperable after 100,000 * 0.1 sec=10,000 sec (2.8 hours). However if you want to hold your profile data such as positioning parameters or any other parameter values that are changed infrequently, you will find this instruction very useful.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
P13	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
EN	BOOL	Activation of the function block (when EN changes from FALSE to TRUE, the function block will be executed one time)
s1	INT, WORD	DT start address of the block(s) that you want to save
s2	DINT, DWORD	Number of blocks to write (1 block = 64 words (DTs))
d	DINT, DWORD	EEPROM start block number
ENO	BOOL	When the function block was executed, ENO is set to TRUE. Helpful at cascading of function blocks with EN–functionality

One of the two input variables s2 or d has to be assigned a constant number value.

Operands

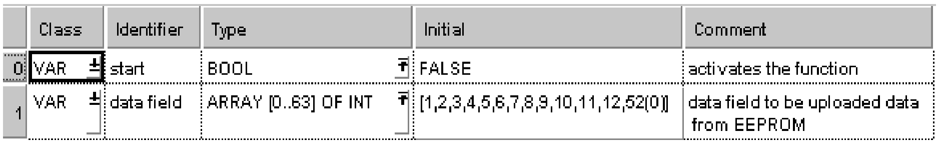
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	–	–	–	–	–	–	x	–	–	–
s2, d	**DWX**	**DWY**	**DWR**	**DWL**	**DSV**	**DEV**	**DDT**	**DLD**	**DFL**	**dec. or hex.**
	x	x	x	–	x	x	x	–	–	x

x: available
–: not available

PLC–specific information

PLC type	FP0 2,7k C10/C14/C16	FP0 5k C32	FP0 10k T32CP
Block size (1 block)	64 words (64x16bit)	64 words (64x16bit)	64 words (64x16bit)
EEPROM start block number	0 to 9	0 to 95	0 to255
Number of blocks to be read / written each execution	1 to 2	1 to 8	1 to 255
Write duration (Additional scan time)	20 ms each block	5 ms each block	5 ms each block
Read duration (Additional scan time)	Less than 1ms each block	less than 1 ms each block	Less than 1ms each block
Max write times	100,000	10,000	10,000
Note: Power down, RUN –> PROG mode changes are also counted			
Max read times	No limit	No limit	No Limit

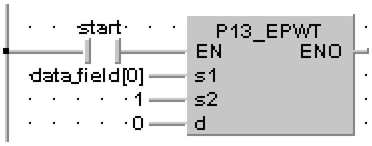
Example In this example the function P13_EPWT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.



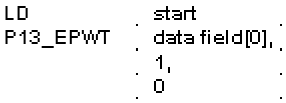
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data field	ARRAY [0..63] OF INT	[1,2,3,4,5,6,7,8,9,10,11,12,52(0)]	data field to be uploaded data from EEPROM

Body When the variable *start* changes from FALSE to TRUE, the function is carried out. The function reads the contents of *data-_field[0]* until *data-_field[63]* (s2* = 1 => 1 block = 64 words) and writes the information after start block number 0 into the EEPROM.

LD



IL



```
LD          start
P13_EPWT    data field[0],
            1,
            0
```

F15_XCH

16–bit data exchange

Steps 5

Description

The contents in the 16–bit areas specified by **d1** and **d2** are exchanged if the trigger **EN** is in the ON–state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F15	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d1	INT, WORD	16–bit area to be exchanged with d2
d2	INT, WORD	16–bit area to be exchanged with d1

The variables **d1** and **d2** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1	–	x	x	x	x	x	x	x	x	–
d2	–	x	x	x	x	x	x	x	x	–

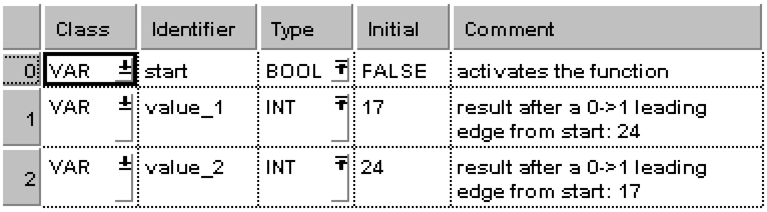
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

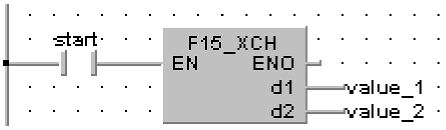


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	INT	17	result after a 0->1 leading edge from start: 24
2	VAR	value_2	INT	24	result after a 0->1 leading edge from start: 17

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F15_XCH(value_1, value_2);
END_IF;
```

F16_DXCH

32–bit data exchange

Steps 5

Description Two 32–bit data specified by **d1** and **d2** are exchanged if the trigger **EN** is in the ON–state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F16	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d1	DINT, DWORD	32–bit area to be exchanged with d2
d2	DINT, DWORD	32–bit area to be exchanged with d1

The variables **d1** and **d2** have to be of the same data type.

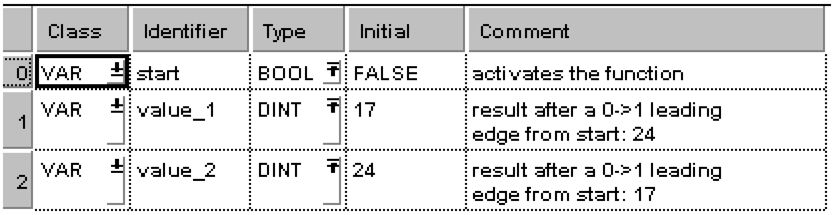
Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d1, d2	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

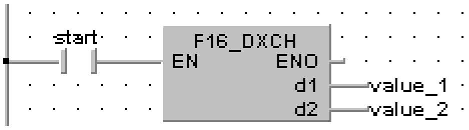
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	DINT	17	result after a 0->1 leading edge from start: 24
2	VAR	value_2	DINT	24	result after a 0->1 leading edge from start: 17

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F16_DXCH(value_1, value_2);
END_IF;
```

F17_SWAP

Higher/lower byte in 16–bit data exchange

Steps	3

Description The higher byte (higher 8 bits) and lower byte (lower 8 bits) of a 16–bit area specified by **d** are exchanged if the trigger **EN** is in the ON–state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F17	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area in which the higher and lower bytes are swapped (exchanged)

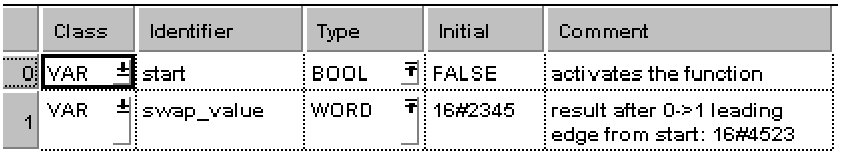
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

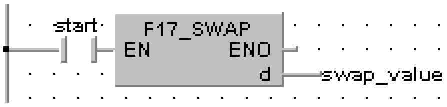
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	swap_value	WORD	16#2345	result after 0->1 leading edge from start: 16#4523

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F17_SWAP(swap_value);
END_IF;
```

F144_TRNS

Serial communication (RS232C)

Steps	5

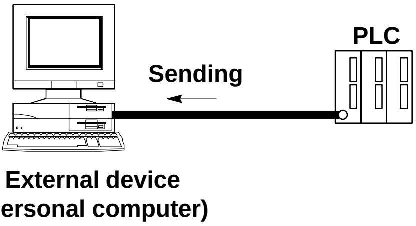
Description

Use this instruction for transmission and reception of command data when an external device (personal computer, measuring instrument, bar code reader, etc.) is connected to the COM. port of the CPU or RS232C port.

Transmission

The **n** bytes of the data stored in the data table with the starting area specified by **s** are transmitted from the COM. port or RS232C port to an external device by serial transmission.

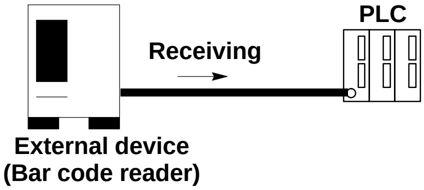
A start code and end code can be automatically added before transmission.



Reception

Reception is controlled by the reception completed flag (R9038) being turned on and off.

When reception completed flag (R9038) is off, the data sent to the COM. port or RS232C port is stored in the reception buffer selected in system registers 417 and 418. When an F144_TRNS instruction is executed, the reception completed flag (R9038) goes off.



PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F144	x	–	x	–	C types

x: available
–: not available

Data types

Variable	Data type	Function
s	WORD	Starting 16-bit area for storing data to be sent.
n	INT, WORD	16-bit equivalent constant or 16-bit area to specify number of bytes to be sent: – When the value is positive, an end code is added. – When the value is negative, no end code is added. – When the value is 16#8000, the transmission mode of the RS232C port is changed from Computer–Link to General purpose or vice–versa.

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	–	–	–	–	–	–	x	–	–	–
n	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the number of bytes specified by n exceeds the source data area range.
R9008	%MX0.900.8	for an instant	

■ Preparation of transmission and reception

1) Setting the use of COM. port: System register 412

F144 is only executed if system register 412 is set to general purpose.

With the programming software:
Set system register 412 for serial transmission (general purpose port).

With the PLC program:
To switch between “computer link communication” and “serial data communication” (general purpose port), execute an **F144_TRNS** instruction. Set **n** (the number of transmission bytes) to 16#8000, and then execute the instruction.

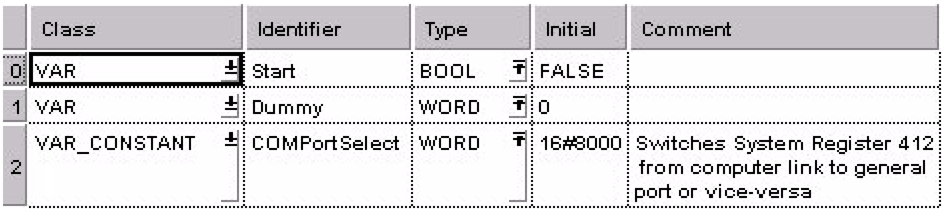
When executed when “computer link” is selected, the setting will change to “general purpose port.”

When executed when “general purpose port” is selected, the setting will change to “computer link.” R9032 is the COM. port selection flag. This flag turns on when “General purpose port” is selected.

Example In this example the function F144_TRNS is programmed in ladder diagram (LD).

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

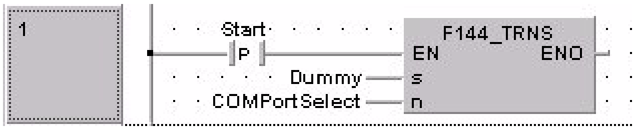


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	Dummy	WORD	0	
2	VAR_CONSTANT	COMPortSelect	WORD	16#8000	Switches System Register 412 from computer link to general port or vice-versa

Body

The variable *ComPortSelect* is assigned the value 16#8000. This means that the COM port setting will switch to general purpose when in computer link mode or vice–versa when the function is executed.

LD



When the power is turned on, the port use will revert to the setting of system register 412.

2) Set the RS232C transmission format with system register 413

The initial settings for the transmission format are as follows:
Data length: 8 bits
Parity check: Yes, odd
Stop bits: 1 bit
End code: C_R
Start code: No STX

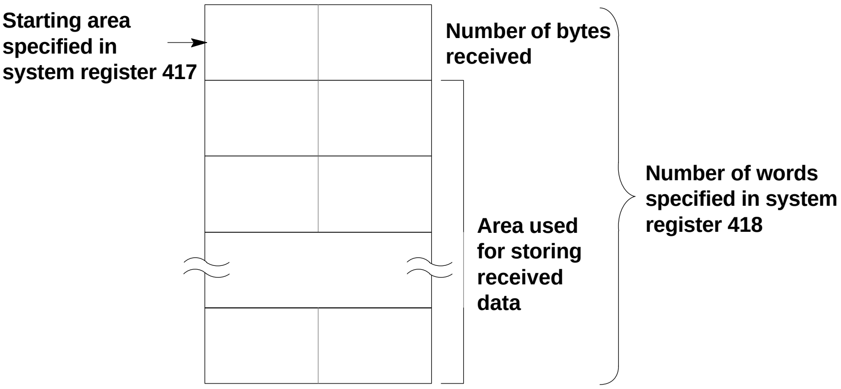
Sets transmission formats according to the connected external device. Since the end code specified in sxstem register 413 is automatically added to data sent, you do not have to write an end code in the area specified by s and n.

3) Set the initial baud rate with system register 414

The baud rate (transmission speed) for serial transmission is initially set to 9600 bps. Sets baud rate of RS232C port according to the connected external device.

4) Setting the reception buffer: System registers 417 and 418

All areas of the data register are initially set for use as the reception buffer. To change the reception buffer, set the starting area number in system register 417 and the size (number of words—maximum of 1000) in system register 418.
The reception buffer will be as follows:


Starting area specified in system register 417
Number of bytes received
Area used for storing received data
Number of words specified in system register 418

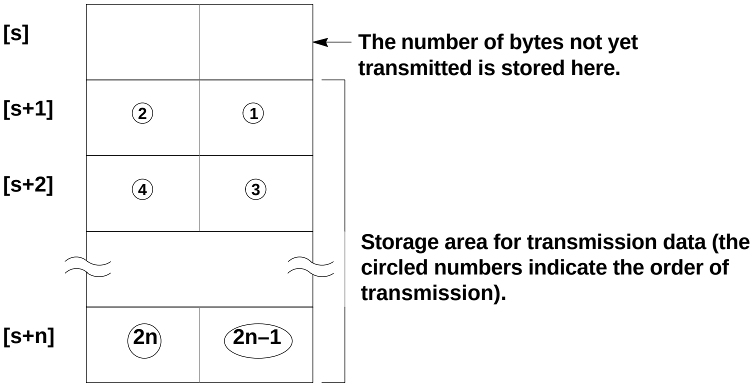
Program and operation during transmission

To transmit, write the transmission data to the data table, select it with an **F144_TRNS** instruction, and execute.

Data table for transmission

Data register areas beginning with the area selected by **s** are used as the data table for transmission.

☞ **Take care that the transmission data table and reception buffer areas (set in system registers 417 and 418) do not overlap.**


[s]
The number of bytes not yet transmitted is stored here.
[s+1]
②
①
[s+2]
④
③
Storage area for transmission data (the circled numbers indicate the order of transmission).
[s+n]
2n
2n–1

Write the transmission data to the transmission data storage area selected with **s** (from the second word on) using an **F0_MV** or **F95_ASC** instruction.

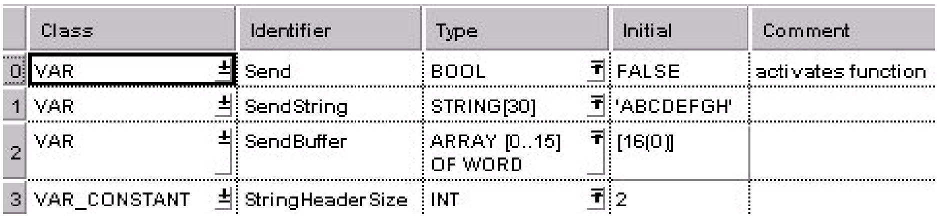
- Do not include an end code in the transmission data as it will be added automatically.
- If the start code is set to “Yes”, do not include a start code in the transmission data as it will be added automatically.
- There is no restriction on the number of bytes **n** that can be transmitted. Following the initial area of the data **s**, transmission is possible up to the data range that can be used by the data register.

When the **F144_TRNS** instruction is executed, the number of data bytes not yet transmitted is stored in the starting area of the data table.

Example In this example the characters of the the string *SendString* are transmitted.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

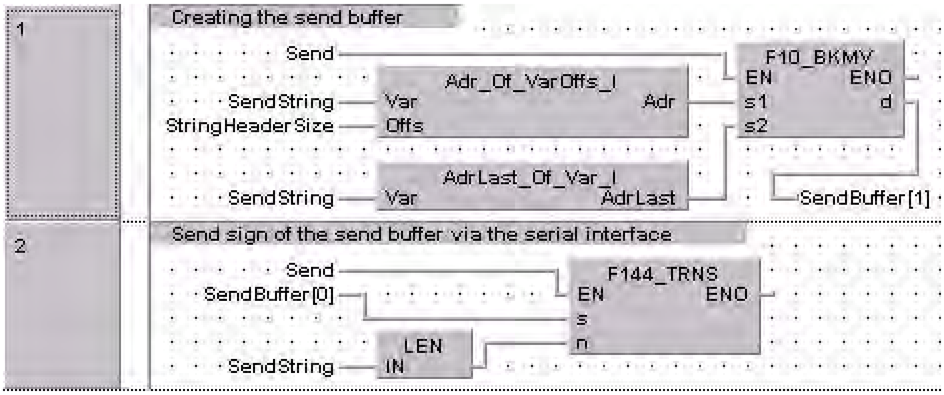


	Class	Identifier	Type	Initial	Comment
0	VAR	Send	BOOL	FALSE	activates function
1	VAR	SendString	STRING[30]	'ABCDEFGH'	
2	VAR	SendBuffer	ARRAY [0..15] OF WORD	[16(0)]	
3	VAR_CONSTANT	StringHeaderSize	INT	2	

Body

When the variable *Send* is set to TRUE, the function F10_BKMV copies the applied data of the string *SendString* to the buffer *SendBuffer* beginning at *SendBuffer[1]*. Additionally, the size of the string header, 2, is added to the beginning address of the string. Two characters of the string *SendString* can be copied into each element of the array *SendBuffer*. *SendBuffer[0]* remains reserved to show the number of bytes to be sent for the instruction F144_TRNS.

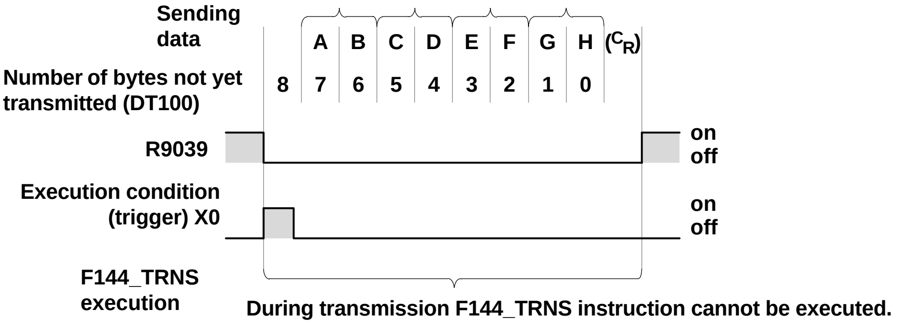
LD



Operation:

If the execution condition (trigger) for the **F144_TRNS** instruction is on when sending completed flag (R9039) goes on, operation will proceed as follows:

1. **n** is preset in **s** (the number of bytes not yet transmitted). Furthermore, reception completed flag (R9038) is turned off and the reception data number is cleared to zero.
2. The data in the data table is transmitted in order from the lower byte.
   - As each byte is transmitted, the value in **s** (the number of bytes not yet transmitted) decrements by 1.
   - During transmission, the sending completed flag (R9039) goes off.
   - If the start code STX is set to “Yes”, the start code will be automatically added to the beginning of the data.
   - The end code selected is automatically added to the end of the data.


Sending data
A B C D E F G H (CR)
Number of bytes not yet transmitted (DT100)
8 7 6 5 4 3 2 1 0
R9039
on
off
Execution condition (trigger) X0
on
off
F144_TRNS execution
During transmission F144_TRNS instruction cannot be executed.

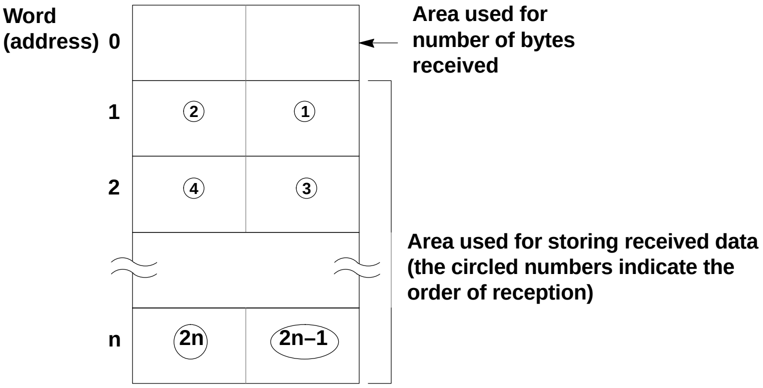
3. When the specified quantity of data has been transmitted, the value in **s** (the number of bytes not yet transmitted) will be zero and the sending completed flag (R9039) will go on.

☞ **The F144_TRNS instruction cannot be executed and the R9039 is not turned on unless pin number 5 of COM. port (RS232C) is turned on.**

■ Program and operation during reception

Data sent from the external device connected to the COM port or RS232C port will be stored in the data register areas set as the reception buffer in system registers 417 and 418.

Reception buffer

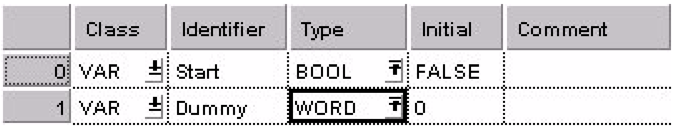

Word (address) 0
1
2
n
② ①
④ ③
2n 2n–1
Area used for number of bytes received
Area used for storing received data (the circled numbers indicate the order of reception)

Each time data is received, the amount of data received (number of bytes) is stored as a count in the leading address of the reception buffer. The initial value is zero.

The data received is stored in order in the reception data storage area beginning from the lower byte of the second word of the area.

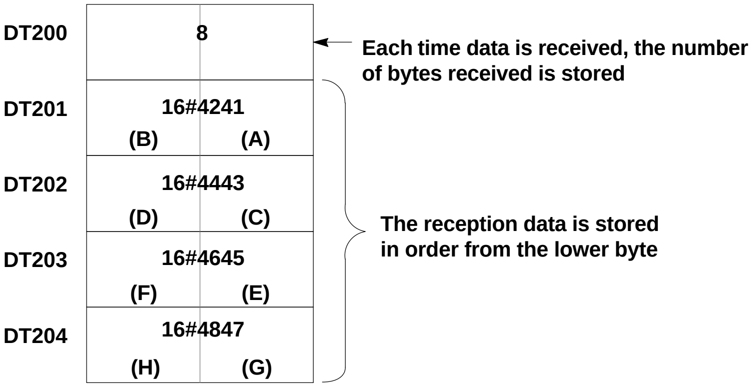
Example In this example the function F144_TRNS is programmed in ladder diagram (LD).

POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	Dummy	WORD	0	

Body In this example, the eight characters A, B, C, D, E, F, G and H (8 bytes of data) are received from an external device.
System register settings for this example are as follows:
– System register 417: 200
– System register 418: 4



DT200	8	
DT201	16#4241 (B)	(A)
DT202	16#4443 (D)	(C)
DT203	16#4645 (F)	(E)
DT204	16#4847 (H)	(G)

Each time data is received, the number of bytes received is stored

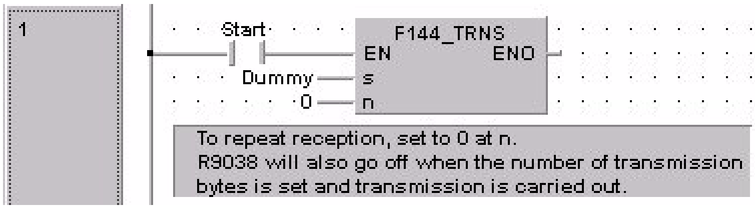
The reception data is stored in order from the lower byte

Reception buffer when reception is completed

When reception of data from an external device has been completed, the reception completed flag (R9038) goes on and further reception of data is not allowed.

To receive more data, an **F144_TRNS** instruction must be executed to turn off the reception completed flag (R9038) and clear the byte number to zero.

LD



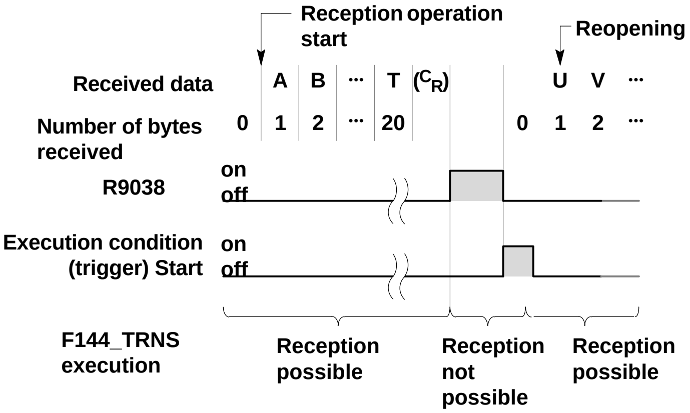
To repeat reception, set to 0 at n.
R9038 will also go off when the number of transmission bytes is set and transmission is carried out.

Operation:
When the reception completed flag (9038) is off and data is sent from an external device, operation will proceed as follows. (After RUN, R9038 is off during the first scan.)

1. The data received is stored in order in the reception data storage area of reception buffer beginning from the lower byte of the second word of the area.

   Start and end codes will not be stored.

   With each one byte received, the value in the leading address of the reception buffer is incremented by 1.



2. When an end code is received, the reception completed flag (R9038) goes on. After this, no further reception of data is allowed.
3. When an **F144_TRNS** instruction is executed, the reception completed flag (R9038) goes off and the number of received data bytes is cleared to zero. Further data received is stored in order in the reception data storage area beginning from the lower byte of the second word of the area.

☞ **For repeated reception of data, refer to the following procedure 1) to 5).**
1) Receive data
2) Reception completed (R9038: on, Reception: not allowed)
3) Process received data
4) Execute F144_TRNS instruction (R9038: off, Reception: enable)
5) Receive further data

F147_PR Parallel printout

Steps	5

Description

Outputs the ASCII codes for 12 characters stored in the 6–word area specified by **s** via the word external output relay specified by **d** if the trigger **EN** is in the ON–state. If a printer is connected to the output specified by **d**, a character corresponding to the output ASCII code is printed.

Only bit positions 0 to 8 of **d** are used in the actual printout. ASCII code is output in sequence starting with the lower byte of the starting area. Three scans are required for 1 character constant output. Therefore, 37 scans are required until all characters constants are output.

Since it is not possible to execute multiple **F147_PR** instructions in one scan, use print–out flag R9033 to be sure they are not executed simultaneously. If the character constants convert to ASCII code, use of the F95_ASC instruction is recommended.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F147	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	starting 16–bit area for storing 12 bytes (6 words) of ASCII codes (source)
d	WORD	word external output relay used for output of ASCII codes (destination)

Operands

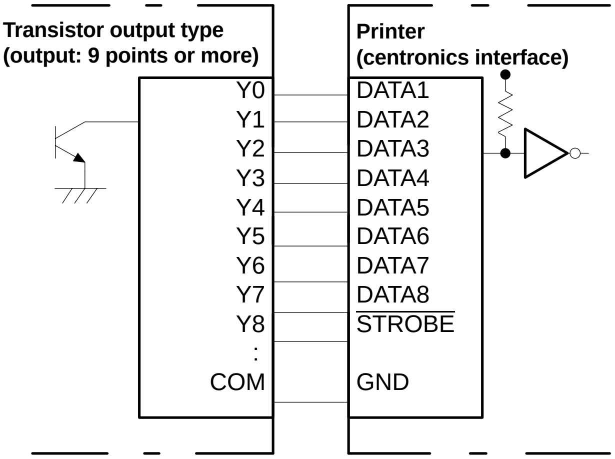
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	–
d	–	x	–	–	–	–	–	–	–	–

x: available
–: not available

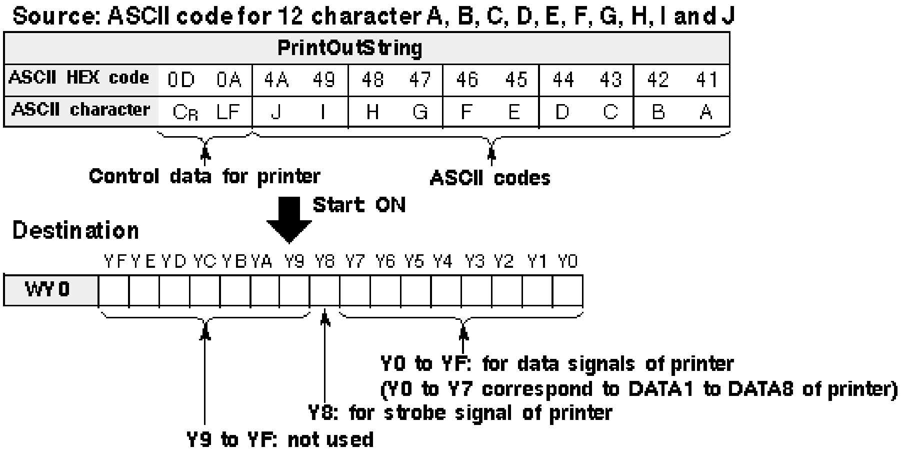
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the ending area for storing ASCII codes exceeds the limit – the trigger of another **F147_PR** instruction turns on while one **F147_PR** instruction is being executed
R9008	%MX0.900.8	for an instant	
R9033	%MX0.903.3	permanently	– a **F147_PR** instruction is being executed

Connection example



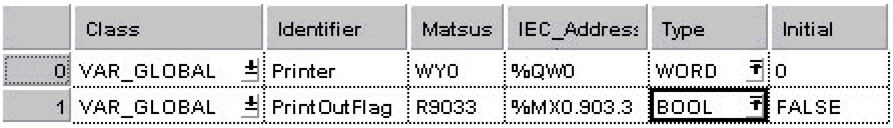
Example In this example the function F147_PR is programmed in ladder diagram (LD). The ASCII codes stored in the string *PrintOutString* are output through word external output relay WY0 when trigger *Start* turns on.


Source: ASCII code for 12 character A, B, C, D, E, F, G, H, I and J

PrintOutString												
ASCII HEX code	0D	0A	4A	49	48	47	46	45	44	43	42	41
ASCII character	CR	LF	J	I	H	G	F	E	D	C	B	A

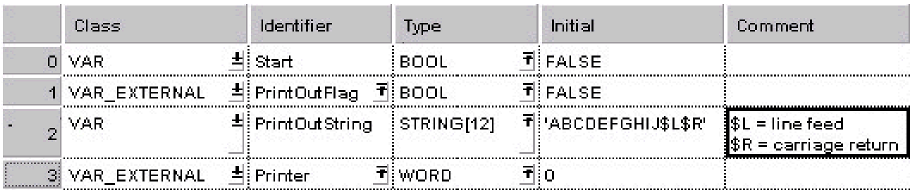
Control data for printer
ASCII codes
Start ON
Destination
YF YE YD YC YB YA Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
WY0
Y0 to YF: for data signals of printer
(Y0 to Y7 correspond to DATA1 to DATA8 of printer)
Y8: for strobe signal of printer
Y9 to YF: not used

GVL In the **G**lobal **V**ariable **L**ist, you define variables that can be accessed by all POUs in the project.



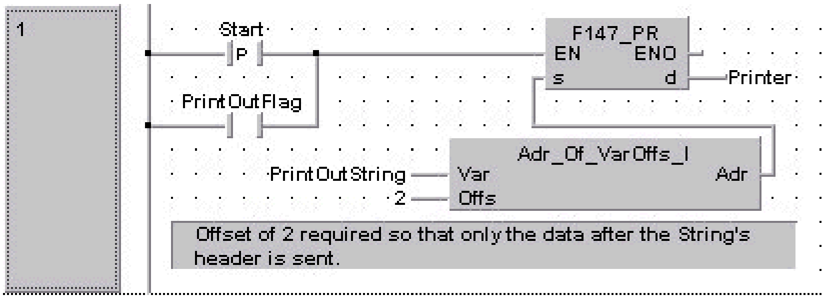
	Class	Identifier	Matsus	IEC_Address	Type	Initial
0	VAR_GLOBAL	Printer	WY0	%QW0	WORD	0
1	VAR_GLOBAL	PrintOutFlag	R9033	%MX0.903.3	BOOL	FALSE

POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR_EXTERNAL	PrintOutFlag	BOOL	FALSE	
2	VAR	PrintOutString	STRING[12]	'ABCDEFGHIJLR'	$L = line feed $R = carriage return
3	VAR_EXTERNAL	Printer	WORD	0	

LD



Offset of 2 required so that only the data after the String's header is sent.

ST

```
IF DF(start) OR PrintOutFlag THEN
      F147_PR(   Adr_Of_VarOffs(   PrintOutString,   2),
Printer);
END_IF;
```

Chapter 16

Arithmetic Instructions

F20_ADD

16–bit addition

Steps	5

Description The 16–bit equivalent constant or 16–bit area specified by **s** and the 16–bit area specified by **d** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F20	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	addend
d	INT, WORD	augend and result

The variables **s** and **d** have to be of the same data type.

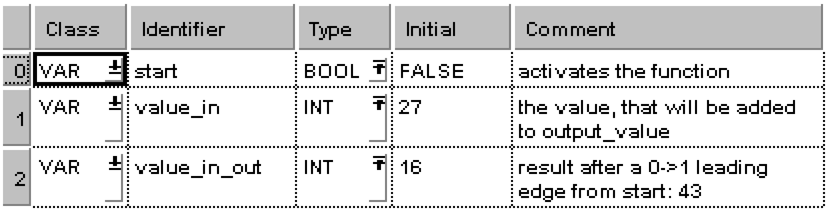
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

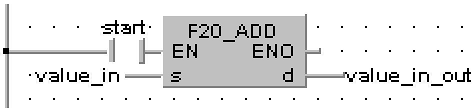
POU header: In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	INT	27	the value, that will be added to output_value
2	VAR	value_in_out	INT	16	result after a 0->1 leading edge from start: 43

Body: When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F20_ADD(value_in, value_in_out);
END_IF;
```

F21_DADD

32–bit addition

Steps	7

Description The 32–bit equivalent constant or 32–bit area specified by **s** and the 32–bit data specified by **d** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F21	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DINT, DWORD	addend
d	DINT, DWORD	augend and result

The variables **s** and **d** have to be of the same data type.

Operands

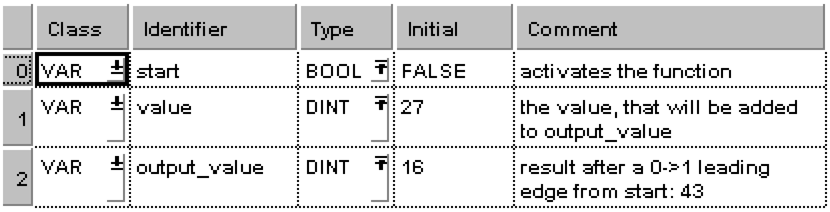
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

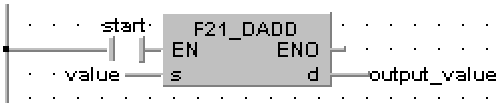


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	DINT	27	the value, that will be added to output_value
2	VAR	output_value	DINT	16	result after a 0->1 leading edge from start: 43

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F21_DADD(value, output_value);
END_IF;
```

F22_ADD2

16–bit addition, destination can be specified

Steps 7

Description

The 16–bit data or 16–bit equivalent constant specified by **s1** and **s2** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F22	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	augend
s2	INT, WORD	addend
d	INT, WORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

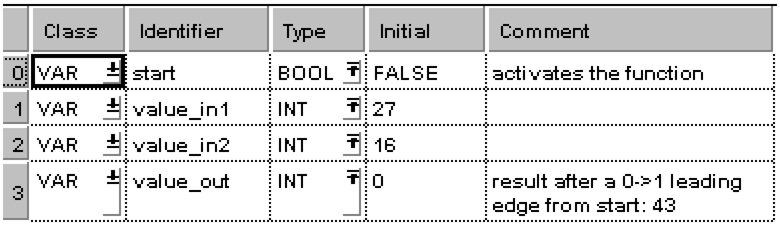
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

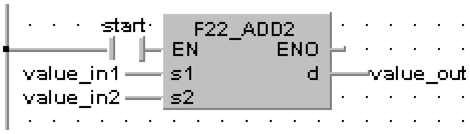


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in1	INT	27	
2	VAR	value_in2	INT	16	
3	VAR	value_out	INT	0	result after a 0->1 leading edge from start: 43

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F22_ADD2(value_in1, value_in2, value_out);
END_IF;
```

F23_DADD2

32–bit addition, destination can be specified

Steps 11

Description The 32–bit data or 32–bit equivalent constant specified by **s1** and **s2** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F23	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DINT, DWORD	augend
s2	DINT, DWORD	addend
d	DINT, DWORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

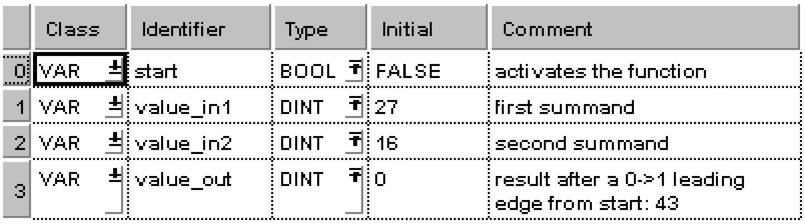
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

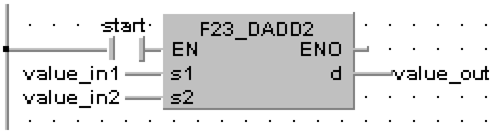


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in1	DINT	27	first summand
2	VAR	value_in2	DINT	16	second summand
3	VAR	value_out	DINT	0	result after a 0->1 leading edge from start: 43

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F23_DADD2(value_in1, value_in2, value_out);
END_IF;
```

F40_BADD

4–digit BCD addition

Steps	5

Description The 4–digit BCD equivalent constant or 16–bit area for 4–digit BCD data specified by **s** and the 16–bit area for 4–digit BCD data specified by **d** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F40	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	WORD	addend, 16–bit area for 4–digit BCD data or equivalent constant
d	WORD	augend and result, 16–bit area for 4–digit BCD data

Operands

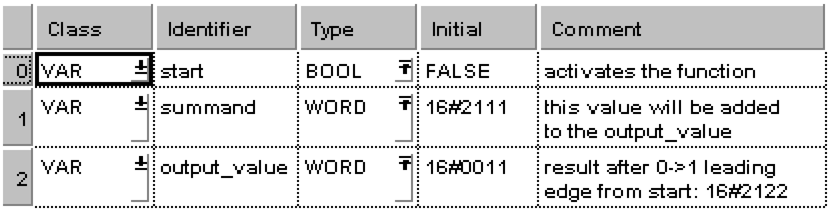
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

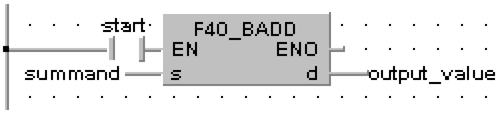


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand	WORD	16#2111	this value will be added to the output_value
2	VAR	output_value	WORD	16#0011	result after 0->1 leading edge from start: 16#2122

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F40_BADD(summand, output_value);
END_IF;
```

F41_DBADD

8–digit BCD addition

Steps 7

Description The 8–digit BCD equivalent constant or 8–digit BCD data specified by **s** and the 8–digit BCD data specified by **d** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F41	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DWORD	addend, 32–bit area for 8–digit BCD data or equivalent constant
d	DWORD	augend and result, 32–bit area for 8–digit BCD data

Operands

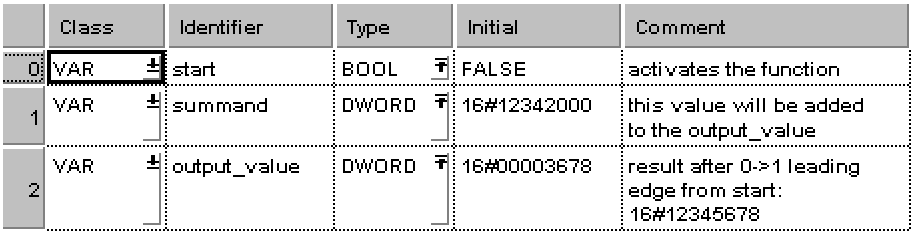
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

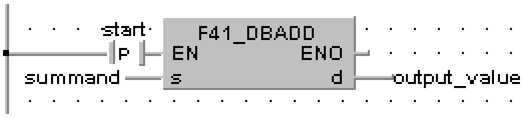


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand	DWORD	16#12342000	this value will be added to the output_value
2	VAR	output_value	DWORD	16#00003678	result after 0->1 leading edge from start: 16#12345678

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F41_DBADD(summand, output_value);
END_IF;
```

F42_BADD2

4–digit BCD addition, destination can be specified

Steps	7

Description The 4–digit BCD equivalent constant or 16–bit area for 4–digit BCD data specified by **s1** and **s2** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F42	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	augend, 16–bit area for 4–digit BCD data or equivalent constant
s2	WORD	addend, 16–bit area for 4–digit BCD data or equivalent constant
d	WORD	sum, 16–bit area for 4–digit BCD data

Operands

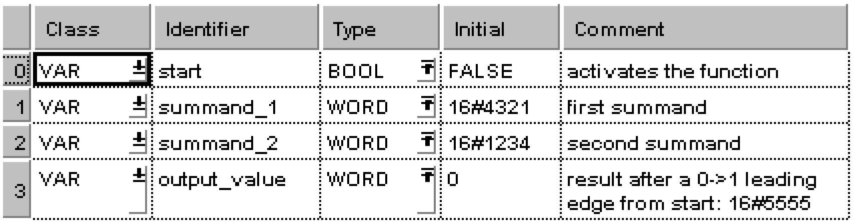
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

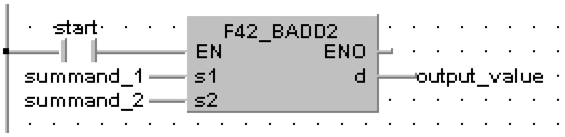


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand_1	WORD	16#4321	first summand
2	VAR	summand_2	WORD	16#1234	second summand
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#5555

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F42_BADD2(summand_1, summand_2, output_value);
END_IF;
```

F43_DBADD2 8–digit BCD addition, destination can be specified

Steps	11

Description The 4–digit BCD equivalent constant or 16–bit area for 4–digit BCD data specified by **s1** and **s2** are added together if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F43	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DWORD	augend, 32–bit area for 8–digit BCD data or equivalent constant
s2	DWORD	addend, 32–bit area for 8–digit BCD data or equivalent constant
d	DWORD	sum, 32–bit area for 8–digit BCD data

Operands

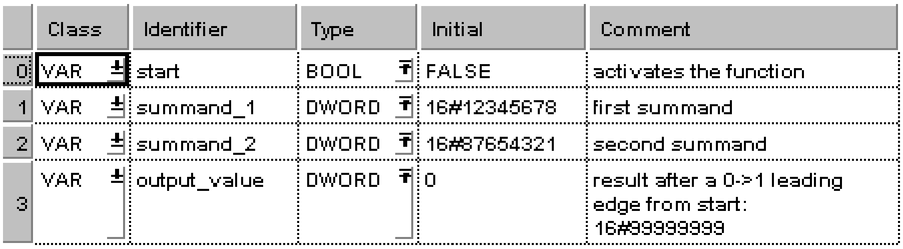
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

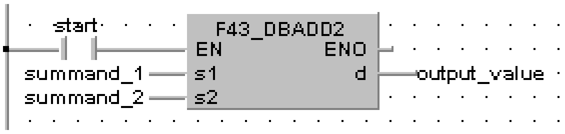


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	summand_1	DWORD	16#12345678	first summand
2	VAR	summand_2	DWORD	16#87654321	second summand
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#99999999

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F43_DBADD2( summand_1, summand_2, output_value);
END_IF;
```

F157_CADD Time addition

Steps 9

Description

The date/clock data (3 words) specified by **s1** and the time data (2 words) specified by **s2** are added together if the trigger **EN** is in the ON–state. The result is stored in the area (3 words, same format as **s1**) specified by **d**. All the data used in the **F157_CADD** instruction are handled in form of BCD.

Example

Clock/calendar data:

August 1, 1992 Time: 14:23:31 (hour:minutes:seconds)

s1[0]: 16#2331 (minutes/seconds)
s1[1]: 16#0114 (day/hour)
s1[2]: 16#9208 (year/month)

Time data:

32 hours; 50 minutes; and 45 seconds

s2: 16#00325045 (hours/minutes/seconds)

You cannot specify special data registers DT9054 to DT9056 (DT90054 to DT90056) for the operand **d**. These registers are factory built–in calendar timer values. To change the built–in calendar timer value, first store the added result in other memory areas and transfer them to the special data registers using the F0_MV instruction.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F157	x	–	5k	–	5k

x: available
–: not available

Data types

Variable	Data type	Function
s1	ARRAY [0..2] OF WORD	augend, time and date, values in BCD format
s2	DWORD	addend, 32–bit area for storing time data in BCD format
d	ARRAY [0..2] OF WORD	sum in BCD format

Operands

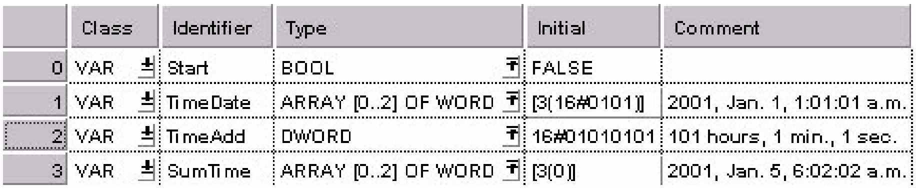
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
d	–	x	x	x	x	x	x	x	x	–
s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function F157_CADD is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

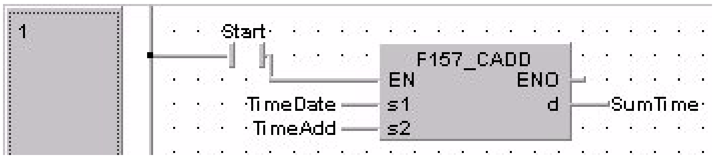
POU header

In the POU header, all input and output variables are declared that are used for programming this function.

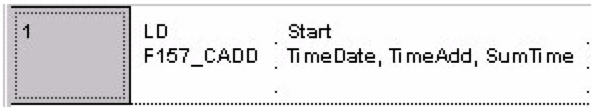


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	TimeDate	ARRAY [0..2] OF WORD	[3(16#0101)]	2001, Jan. 1, 1:01:01 a.m.
2	VAR	TimeAdd	DWORD	16#01010101	101 hours, 1 min., 1 sec.
3	VAR	SumTime	ARRAY [0..2] OF WORD	[3(0)]	2001, Jan. 5, 6:02:02 a.m.

LD



IL



```
1   LD          Start
    F157_CADD   TimeDate, TimeAdd, SumTime
```

F25_SUB

16–bit subtraction

Steps	5

Description

Subtracts the 16–bit equivalent constant or 16–bit area specified by **s** from the 16–bit area specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d** (minuend area).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F25	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	subtrahend
d	INT, WORD	minuend and result

The variables **s** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

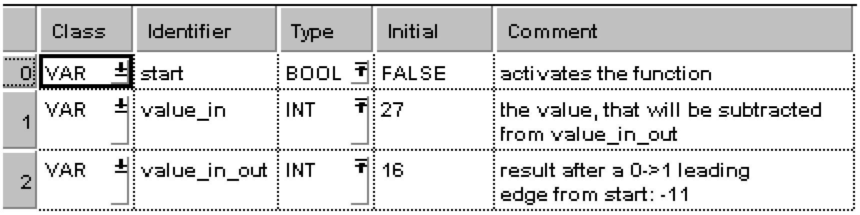
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

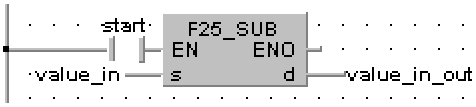


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	INT	27	the value, that will be subtracted from value_in_out
2	VAR	value_in_out	INT	16	result after a 0->1 leading edge from start: -11

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F25_SUB(value_in, value_in_out);
END_IF;
```

F26_DSUB

32–bit subtraction

Steps	7

Description Subtracts the 32–bit equivalent constant or 32–bit data specified by **s** from the 32–bit data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d** (minuend area).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F26	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DINT, DWORD	subtrahend
d	DINT, DWORD	minuend and result

The variables **s** and **d** have to be of the same data type.

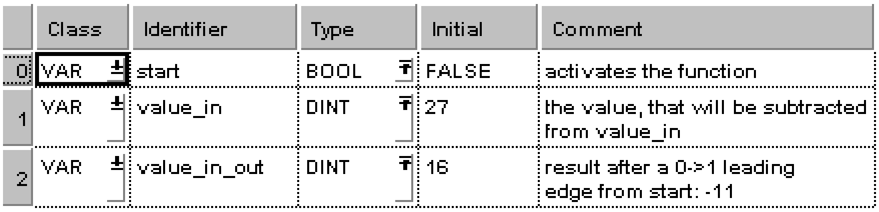
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

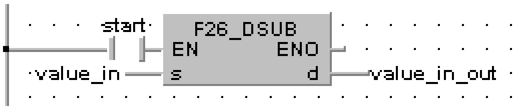
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_in	DINT	27	the value, that will be subtracted from value_in
2	VAR	value_in_out	DINT	16	result after a 0->1 leading edge from start: -11

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F26_DSUB(value_in, value_in_out);
END_IF;
```

F27_SUB2

16–bit subtraction, destination can be specified

Steps	7

Description Subtracts the 16–bit data or 16–bit equivalent constant specified by **s2** from the 16–bit data or 16–bit equivalent constant specified by **s1** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F27	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	minuend
s2	INT, WORD	subtrahend
d	INT, WORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

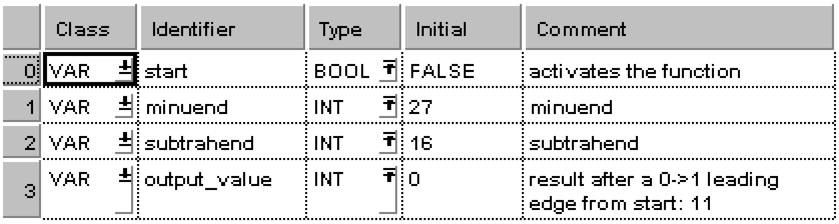
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

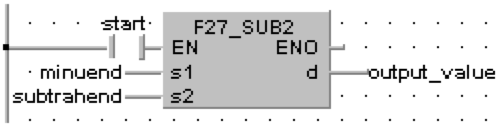


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	INT	27	minuend
2	VAR	subtrahend	INT	16	subtrahend
3	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 11

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F27_SUB2(minuend, subtrahend, output_value);
END_IF;
```

F28_DSUB2

32–bit subtraction, destination can be specified

Steps 11

Description Subtracts the 32–bit data or 32–bit equivalent constant specified by **s2** from the 32–bit data or 32–bit equivalent constant specified by **s1** if the trigger is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F28	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DINT, DWORD	minuend
s2	DINT, DWORD	subtrahend
d	DINT, DWORD	result

The variables **s1, s2** and **d** have to be of the same data type.

Operands

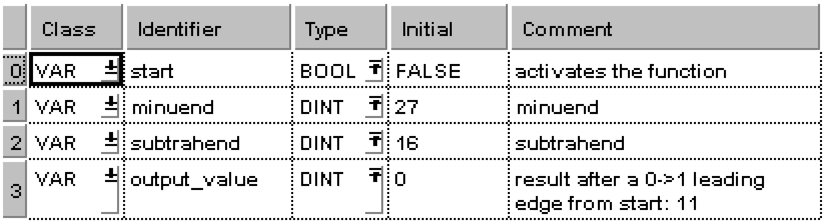
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

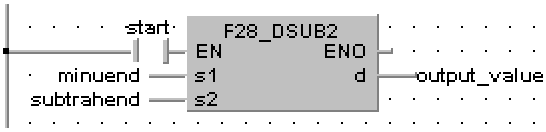


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	DINT	27	minuend
2	VAR	subtrahend	DINT	16	subtrahend
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 11

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F28_DSUB2(minuend, subtrahend, output_value);
END_IF;
```

F45_BSUB

4–digit BCD subtraction

Steps	5

Description Subtracts the 4–digit BCD equivalent constant or 16–bit area for 4–digit BCD data specified by **s** from the 16–bit area for 4–digit BCD data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F45	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	WORD	subtrahend, 16–bit area for 4–digit BCD data or equivalent constant
d	WORD	minuend and result, 16–bit area for 4–digit BCD data

Operands

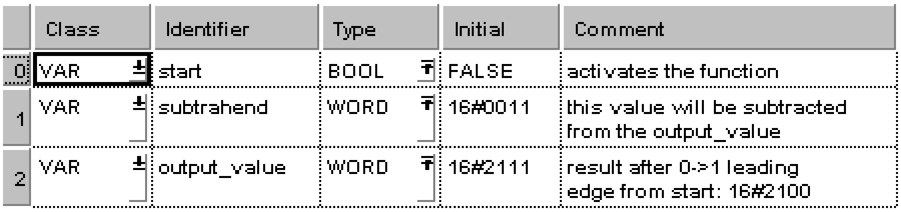
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

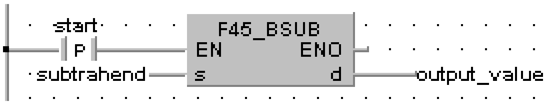


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	subtrahend	WORD	16#0011	this value will be subtracted from the output_value
2	VAR	output_value	WORD	16#2111	result after 0->1 leading edge from start: 16#2100

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F45_BSUB(subtrahend, output_value);
END_IF;
```

F46_DBSUB

8–digit BCD subtraction

Steps	7

Description Subtracts the 8–digit BCD equivalent constant or 8–digit BCD data specified by **s** from the 8–digit BCD data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F46	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DWORD	subtrahend, 32–bit area for 8–digit BCD data or equivalent constant
d	DWORD	minuend and result, 32–bit area for 8–digit BCD data

Operands

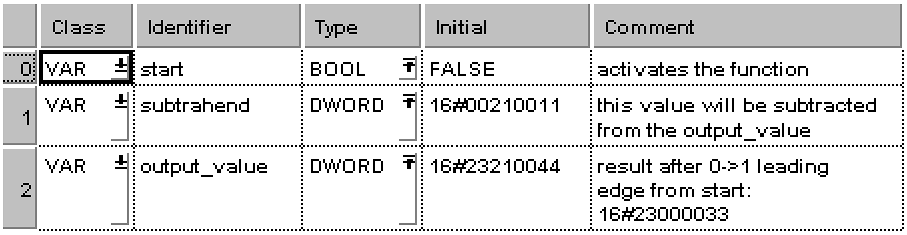
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

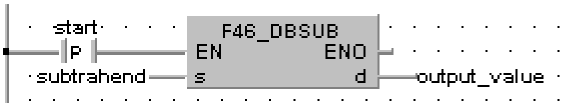


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	subtrahend	DWORD	16#00210011	this value will be subtracted from the output_value
2	VAR	output_value	DWORD	16#23210044	result after 0->1 leading edge from start: 16#23000033

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F46_DBSUB(subtrahend, output_value);
END_IF;
```

F47_BSUB2

4–digit BCD subtraction, destination can be specified

Steps	7

Description Subtracts the 4–digit BCD equivalent constant or 16–bit area for 4–digit BCD data specified by **s2** from the 4–digit BCD equivalent constant or 16–bit area for 4–digit BCD data specified by **s1** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F47	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	minuend, 16–bit area for 4–digit BCD data or equivalent constant
s2	WORD	subtrahend, 16–bit area for 4–digit BCD data or equivalent constant
d	WORD	result, 16–bit area for 4–digit BCD data

Operands

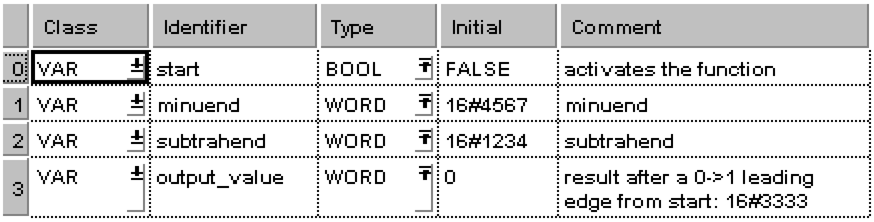
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

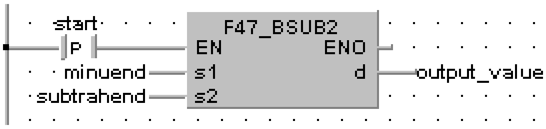


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	WORD	16#4567	minuend
2	VAR	subtrahend	WORD	16#1234	subtrahend
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 16#3333

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F47_BSUB2(minuend, subtrahend, output_value);
END_IF;
```

F48_DBSUB2 8–digit BCD subtraction, destination can be specified

Steps	11

Description Subtracts the 8–digit BCD equivalent constant or 8–digit BCD data specified by **s2** from the 8–digit BCD equivalent constant or 8–digit BCD data specified by **s1** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F48	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DWORD	minuend, 32–bit area for 8–digit BCD data or equivalent constant
s2	DWORD	subtrahend, 32–bit area for 8–digit BCD data or equivalent constant
d	DWORD	result, 32–bit area for 8–digit BCD data

Operands

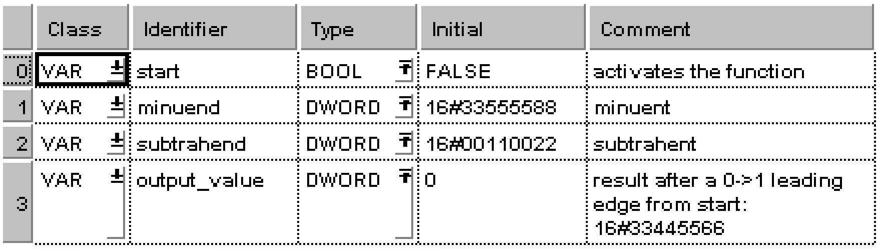
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

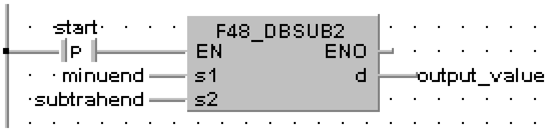


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	minuend	DWORD	16#33555588	minuent
2	VAR	subtrahend	DWORD	16#00110022	subtrahent
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start: 16#33445566

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F48_DBSUB2(minuend, subtrahend, output_value);
END_IF;
```

F158_CSUB Time subtraction

Steps	9

Description Subtracts time data (2 words) specified by **s2** from the date/clock data (3 words) specified by **s1** if the trigger **EN** is in the ON–state. The result is stored in the area (3 words, same format than **s1**) specified by **d**. All the data used in the **F158_CSUB** instruction are handled in form of BCD.

Example **Clock/calendar data:**

August 1, 1992 Time: 14:23:31 (hour:minutes:seconds)

s1[0]: 16#2331 (minutes/seconds)
s1[1]: 16#0114 (day/hour)
s1[2]: 16#9208 (year/month)

Time data:

32 hours; 50 minutes; and 45 seconds

s2 16#00325045 (hours/minutes/seconds)

You cannot specify special data registers DT9054 to DT9056 (DT90054 to DT90056 for FP10/10S) for the operand **d.** These registers factory built–in calendar timer values. To change the built–in calendar timer value, first store the added result in other memory areas and transfer them to the special data registers using the F0_MV instruction.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F158	x	–	5k	–	5k

x: available
–: not available

Data types

Variable	Data type	Function
s1	ARRAY [0..2] OF WORD	minuend, time and date, values in BCD format
s2	DWORD	subtrahend, 32–bit area for storing time data in BCD format
d	ARRAY [0..2] OF WORD	result in BCD format

Operands

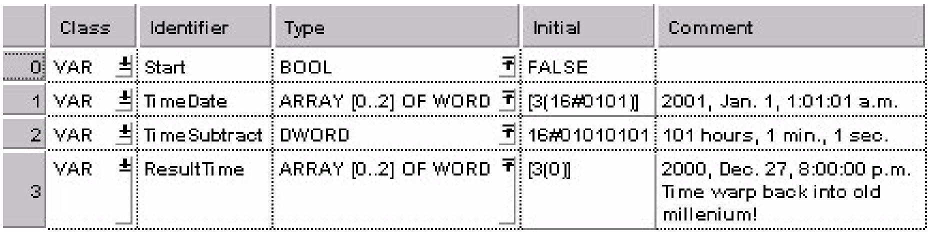
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
d	–	x	x	x	x	x	x	x	x	–
s2	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function F158_CSUB is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

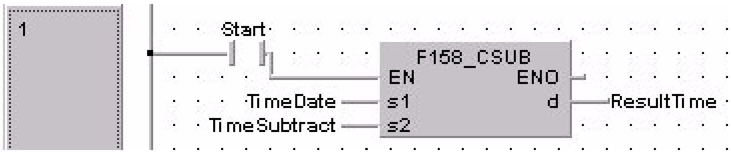
POU header

In the POU header, all input and output variables are declared that are used for programming this function.

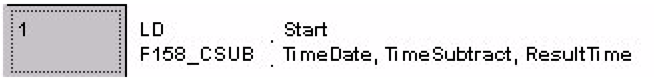


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	TimeDate	ARRAY [0..2] OF WORD	[3(16#0101)]	2001, Jan. 1, 1:01:01 a.m.
2	VAR	TimeSubtract	DWORD	16#01010101	101 hours, 1 min., 1 sec.
3	VAR	ResultTime	ARRAY [0..2] OF WORD	[3(0)]	2000, Dec. 27, 8:00:00 p.m. Time warp back into old millenium!

LD



IL



```
1   LD           Start
    F158_CSUB    TimeDate, TimeSubtract, ResultTime
```

F30_MUL

16–bit multiplication, destination can be specified

Steps	7

Description Multiplies the 16–bit data or 16–bit equivalent constant **s1** and the 16–bit data or 16–bit equivalent constant specified by **s2** if the trigger **EN** is in the ON–state. The result is stored in **d** (32–bit area).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F30	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	multiplicand
s2	INT, WORD	multiplier
d	DINT, DWORD	result

The variables **s1**, **s2** and **d** have to be of the same data type (INT/DINT or WORD/DWORD).

Operands

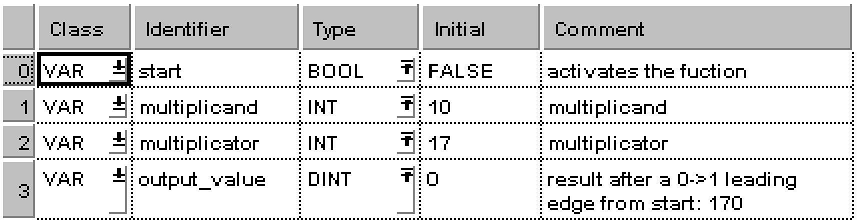
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



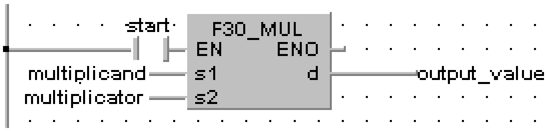
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	multiplicand	INT	10	multiplicand
2	VAR	multiplicator	INT	17	multiplicator
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 170

In this example the input variables *input_value_1, input_value _2* and *input_value _3* are declared. However, you can write constants directly at the input contact of the function instead.

Body

When the variable *start* is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
      F30_MUL(multiplicand, multiplicator, output_value);
END_IF;
```

F31_DMUL

32–bit multiplication, destination can be specified

Steps	11

Description

Multiplies the 32–bit data or 32–bit equivalent constant specified by **s1** and the one specified by **s2** if the trigger **EN** is in the ON–state. The result is stored in **d[1]**, **d[2]** (64–bit area).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F31	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DINT, DWORD	multiplicand
s2	DINT, DWORD	multiplier
d	ARRAY [0..1] OF DINT or DWORD	result

The variables **s1**, **s2** and **d** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

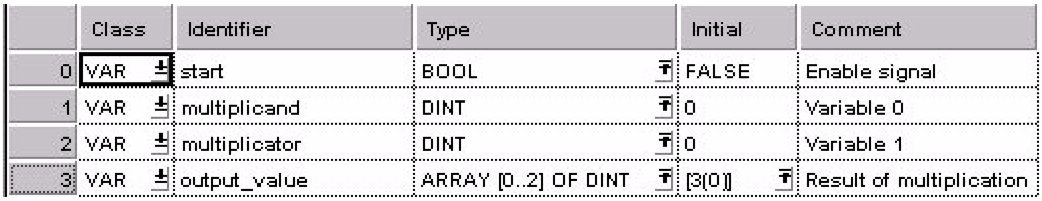
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

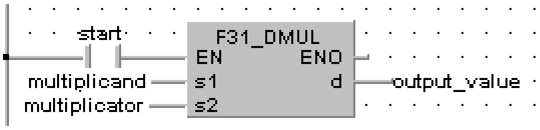


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	Enable signal
1	VAR	multiplicand	DINT	0	Variable 0
2	VAR	multiplicator	DINT	0	Variable 1
3	VAR	output_value	ARRAY [0..2] OF DINT	[3(0)]	Result of multiplication

Body

When the variable *start* is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
      F31_DMUL(multiplicand, multiplicator, output_value);
END_IF;
```

F50_BMUL

4–digit BCD multiplication, destination can be specified

Steps 7

Description Multiplies the 4–digit BCD equivalent constant or 16–bit area for 4–digit BCD data specified by **s1** and **s2** if the trigger **EN** is in the ON–state. The result is stored in **d** (8–digit area).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F50	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	multiplicand, 16–bit area for 4–digit BCD data or equivalent constant
s2	WORD	multiplier, 16–bit area for 4–digit BCD data or equivalent constant
d	DWORD	result, 32–bit area for 8–digit BCD data

Operands

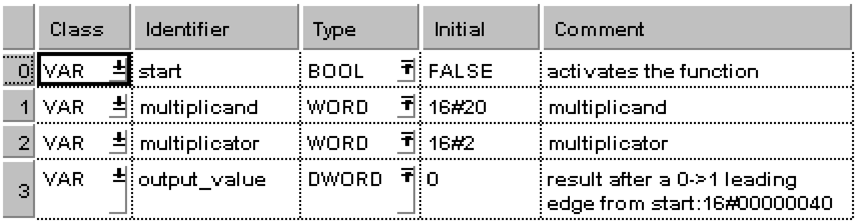
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

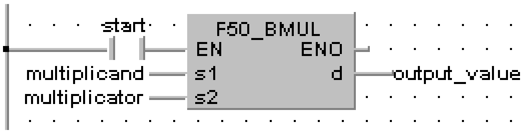


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	WORD	16#20	multiplicand
2	VAR	multiplicator	WORD	16#2	multiplicator
3	VAR	output_value	DWORD	0	result after a 0->1 leading edge from start:16#00000040

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F50_BMUL(multiplicand, multiplicator, output_value);
END_IF;
```

F51_DBMUL

8–digit BCD multiplication, destination can be specified

Steps	11

Description Multiplies the 8–digit BCD equivalent constant or 8–digit BCD data specified by **s1** and the one specified by **s2** if the trigger **EN** is in the ON–state. The result is stored in the ARRAY **d[1]**, **d[2]** (64–digit area).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F51	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DWORD	multiplicand, 32–bit area for 8–digit BCD data or equivalent constant
s2	DWORD	multiplier, 32–bit area for 8–digit BCD data or equivalent constant
d	ARRAY [0..1] OF DWORD	result

Operands

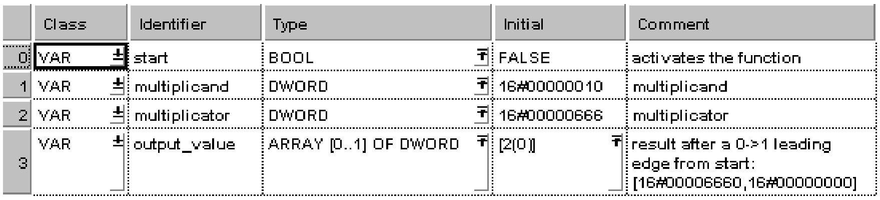
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

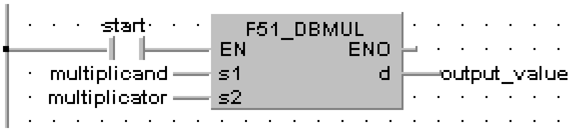


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	multiplicand	DWORD	16#00000010	multiplicand
2	VAR	multiplicator	DWORD	16#00000666	multiplicator
3	VAR	output_value	ARRAY [0..1] OF DWORD	[2(0)]	result after a 0->1 leading edge from start: [16#00006660,16#00000000]

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
     F51_DBMUL(multiplicand, multiplicator, output_value);
END_IF;
```

F32_DIV

16–bit division, destination can be specified

Steps	7

Description

The 16–bit data or 16–bit equivalent constant specified by **s1** is divided by the 16–bit data or 16–bit equivalent constant specified by **s2** if the trigger **EN** is in the ON–state. The quotient is stored in **d** and the remainder is stored in the special data register DT9015/DT90015.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F32	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	dividend
s2	INT, WORD	divisor
d	INT, WORD	quotient

The variables **s1**, **s2** and **d** have to be of the same data type.

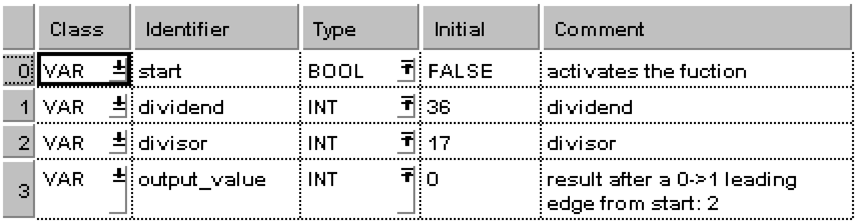
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

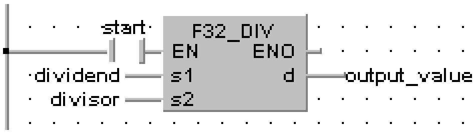
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	dividend	INT	36	dividend
2	VAR	divisor	INT	17	divisor
3	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 2

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F32_DIV(dividend, divisor, output_value);
END_IF;
```

F33_DDIV

32–bit division, destination can be specified

Steps	11

Description

The 32–bit data or 32–bit equivalent constant specified by **s1** is divided by the 32–bit data or 32–bit equivalent constant specified by **s2** if the trigger **EN** is in the ON–state. The quotient is stored in **d** and the remainder is stored in the special data registers DT9016 and DT9015/DT90016 and DT90015.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F33	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DINT, DWORD	dividend
s2	DINT, DWORD	divisor
d	DINT, DWORD	quotient

The variables **s1**, **s2** and **d** have to be of the same data type.

Operands

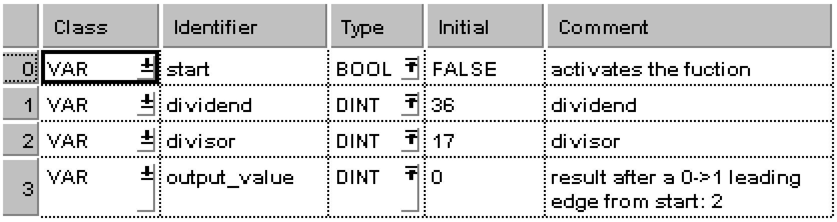
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

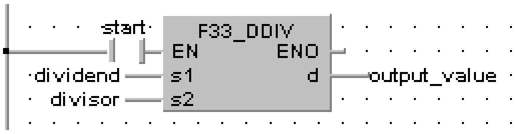


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	dividend	DINT	36	dividend
2	VAR	divisor	DINT	17	divisor
3	VAR	output_value	DINT	0	result after a 0->1 leading edge from start: 2

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F33_DDIV(dividend, divisor, output_value);
END_IF;
```

F52_BDIV

4–digit BCD division, destination can be specified

Steps	7

Description The 4–digit BCD equivalent constant or the 16–bit area for 4–digit BCD data specified by **s1** is divided by the 4–digit BCD equivalent constant or the 16–bit area for 4–digit BCD data specified by **s2** if the trigger **EN** is in the ON–state. The quotient is stored in the area specified by **d** and the remainder is stored in special data register DT9015/DT90015.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F52	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	dividend, 16–bit area for BCD data or 4–digit BCD equivalent constant
s2	WORD	divisor, 16–bit area for BCD data or 4–digit BCD equivalent constant
d	WORD	quotient, 16–bit area for BCD data (remainder stored in special data register DT9015/DT90015)

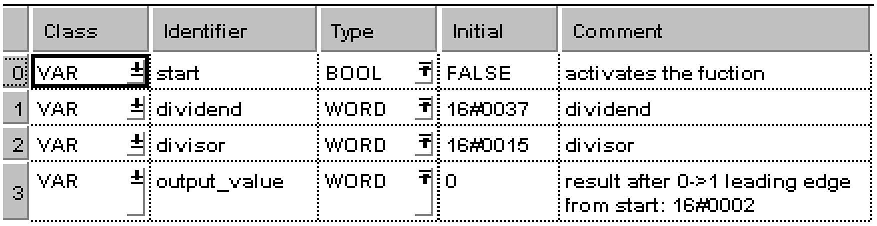
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

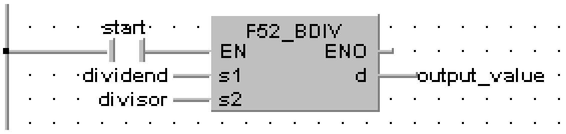
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	dividend	WORD	16#0037	dividend
2	VAR	divisor	WORD	16#0015	divisor
3	VAR	output_value	WORD	0	result after 0->1 leading edge from start: 16#0002

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F52_BDIV(dividend, divisor, output_value);
END_IF;
```

F53_DBDIV

8–digit BCD division, destination can be specified

Steps	11

Description The 8–digit BCD equivalent constant or the 8–digit BCD data specified by **s1** is divided by the 8–digit BCD equivalent constant or the 8–digit BCD data specified by **s2** if the trigger **EN** is in the ON–state. The result is stored in the areas specified by **d**, and the remainder is stored in the special data registers DT9016 and DT9015/DT90016 and DT90015.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F53	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DWORD	dividend, 32–bit area for BCD data or 8–digit BCD equivalent constant
s2	DWORD	divisor, 32–bit area for BCD data or 8–digit BCD equivalent constant
d	DWORD	quotient, 32–bit area for BCD data (remainder stored in special data register DT9016 and DT9015/ DT90016 and DT90015)

Operands

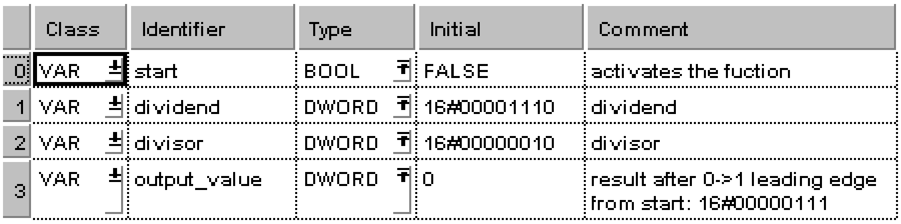
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

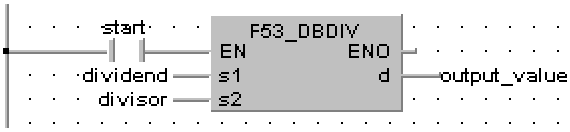


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	dividend	DWORD	16#00001110	dividend
2	VAR	divisor	DWORD	16#00000010	divisor
3	VAR	output_value	DWORD	0	result after 0->1 leading edge from start: 16#00000111

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F53_DBDIV(dividend, divisor, output_value);
END_IF;
```

F35_INC

16–bit increment

Steps 3

Description Adds ”1” to the 16–bit data specified by **d** if the trigger **EN** is in the ON–state. The added result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F35	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area to be increased by 1

Operands

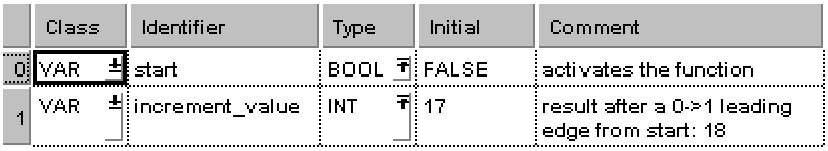
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

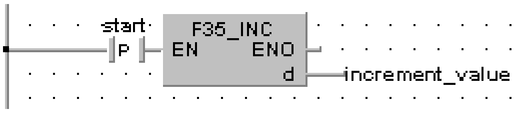


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	INT	17	result after a 0->1 leading edge from start: 18

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F35_INC(increment_value);
END_IF;
```

F36_DINC

32–bit increment

Steps	3

Description Adds ”1” to the 32–bit data specified by **d** if the trigger **EN** is in the ON–state. The added result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F36	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	DINT, DWORD	32–bit area to be increased by 1

Operands

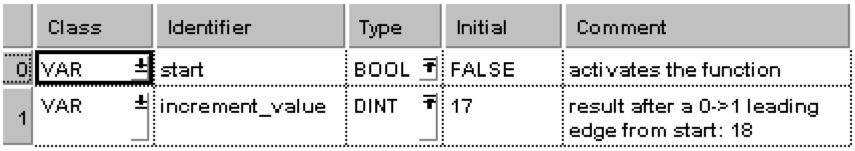
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

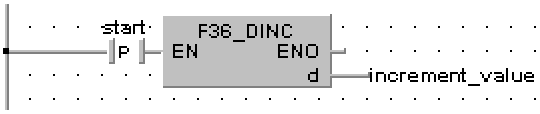


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	DINT	17	result after a 0->1 leading edge from start: 18

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F36_DINC(increment_value);
END_IF;
```

F55_BINC

4–digit BCD increment

Steps	3

Description Adds ”1” to the 4–digit BCD data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F55	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
d	WORD	16–bit area for 4–digit BCD data to be increased by 1

Operands

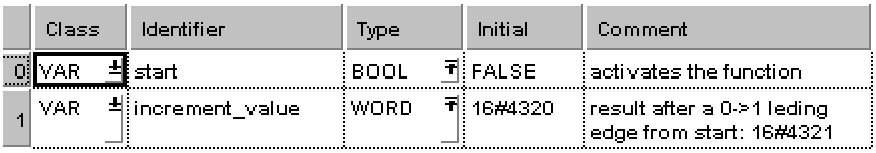
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

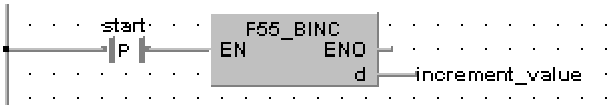


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	WORD	16#4320	result after a 0->1 leding edge from start: 16#4321

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F56_DBINC(increment_value);
END_IF;
```

F56_DBINC 8–digit BCD increment

Steps	3

Description Adds ”1” to the 8–digit BCD data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F56	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
d	DWORD	32–bit area for 8–digit BCD data to be increased by 1

Operands

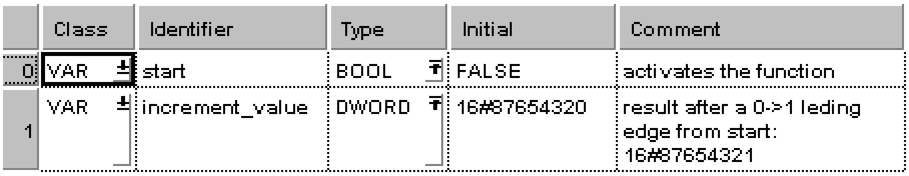
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

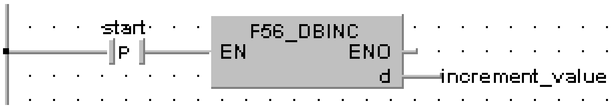


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	increment_value	DWORD	16#87654320	result after a 0->1 leding edge from start: 16#87654321

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F56_DBINC(increment_value);
END_IF;
```

F37_DEC

16–bit decrement

Steps	3

Description Subtracts ”1” from the 16–bit data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F37	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area to be decreased by 1

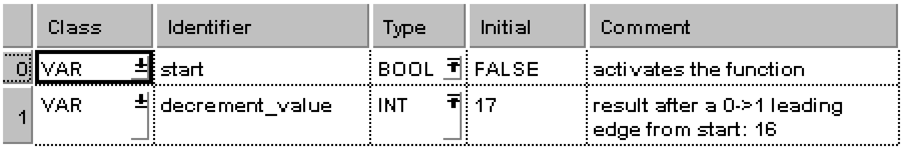
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

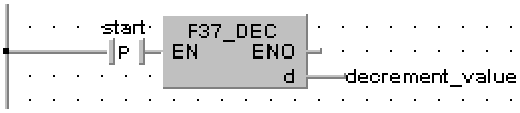
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	INT	17	result after a 0->1 leading edge from start: 16

Body When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F56_DBINC(increment_value);
END_IF;
```

F38_DDEC

32–bit decrement

Steps	3

Description Subtracts ”1” to the 32–bit data specified by **d** if the trigger **EN** is in the ON–state. The added result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F38	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	DINT, DWORD	32–bit area to be decreased by 1

Operands

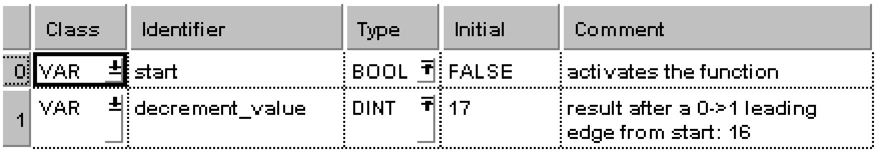
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

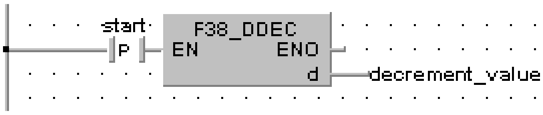


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	DINT	17	result after a 0->1 leading edge from start: 16

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F57_BDEC(decrement_value);
END_IF;
```

F57_BDEC

4–digit BCD decrement

Steps	3

Description Subtracts ”1” from the 4–digit BCD data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F57	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
d	WORD	16–bit area for BCD data to be decreased by 1

Operands

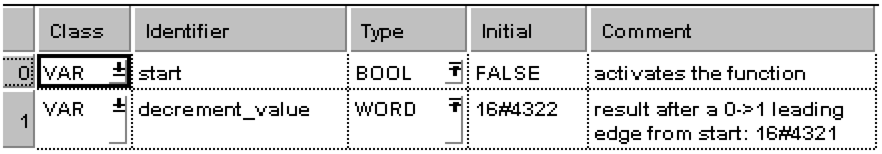
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

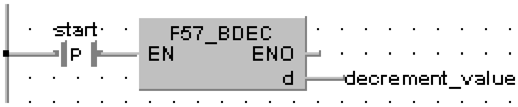


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	WORD	16#4322	result after a 0->1 leading edge from start: 16#4321

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F57_BDEC(decrement_value);
END_IF;
```

F58_DBDEC 8–digit BCD decrement

Steps	3

Description Subtracts ”1” from the 8–digit BCD data specified by **d** if the trigger **EN** is in the ON–state. The result is stored in **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F58	x	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
d	DWORD	32–bit area for BCD data to be decreased by 1

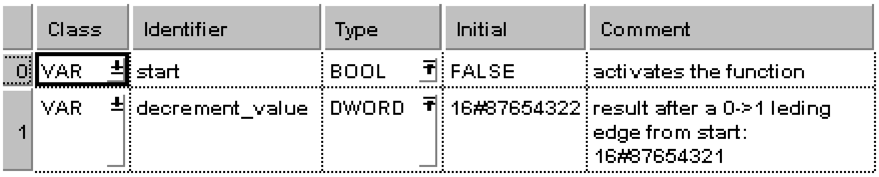
Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

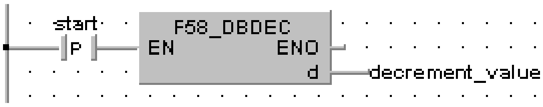
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	decrement_value	DWORD	16#87654322	result after a 0->1 leding edge from start: 16#87654321

Body When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F58_DBDEC(decrement_value);
END_IF;
```

F87_ABS

16–bit data absolute value

Steps	3

Description Gets the absolute value of 16–bit data with the sign specified by **d** if the trigger **EN** is in the ON–state. The absolute value of the 16–bit data with +/– sign is stored in **d**. This instruction is useful to operate the data whose sign (+/–) may vary.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F87	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area for storing original data and its absolute value

Operands

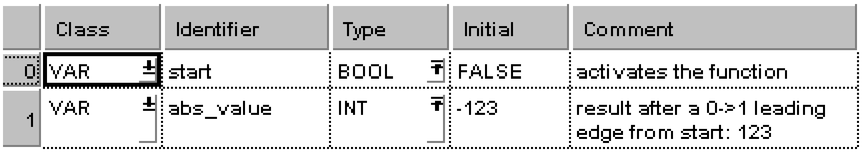
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

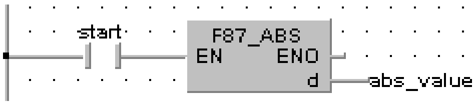


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	abs_value	INT	-123	result after a 0->1 leading edge from start: 123

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F87_ABS(abs_value);
END_IF;
```

F88_DABS

32–bit data absolute value

Steps	3

Description Gets the absolute value of 32–bit data with the sign specified by **d** if the trigger **EN** is in the ON–state. The absolute value of the 32–bit data with sign is stored in **d**. This instruction is useful to operate the data whose sign (+/–) may vary.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F88	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	DINT, DWORD	32–bit area for storing original data and its absolute value

Operands

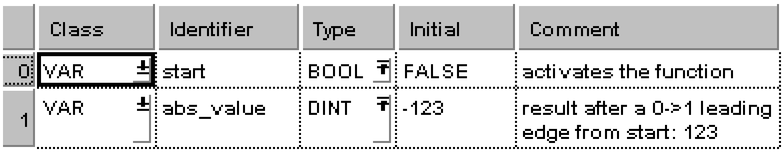
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

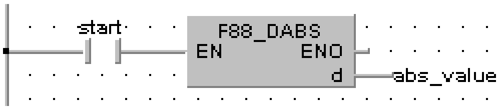


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	abs_value	DINT	-123	result after a 0->1 leading edge from start: 123

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F88_DABS(abs_value);
END_IF;
```

Chapter 17

Data Comparison Instructions

F60_CMP

16–bit data compare

Steps	5

Description Compares the 16–bit data specified by **s1** with one specified by **s2** if the trigger **EN** is in the ON–state. The compare operation result is stored in special internal relays R9009, R900A to R900C.

Data	comparison between s1 and s2	Flag			
		R900A (>flag)	R900B (=flag)	R900C (<flag)	R9009 (carry–flag)
16–bit data with sign	s1< s2	OFF	OFF	ON	#
	s1=s2	OFF	ON	OFF	OFF
	s1> s2	ON	OFF	OFF	#
16–bit data without sign	s1< s2	#	OFF	#	ON
	s1= s2	OFF	ON	OFF	OFF
	s1> s2	#	OFF	#	OFF

#: turns ON or OFF depending on the conditions

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F60	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit area or 16–bit equivalent constant to be compared
s2	INT, WORD	16–bit area or 16–bit equivalent constant to be compared

The variables **s1** and **s2** have to be of the same data type.

Operands

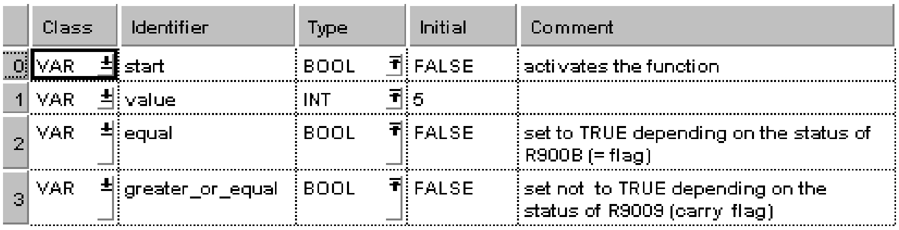
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

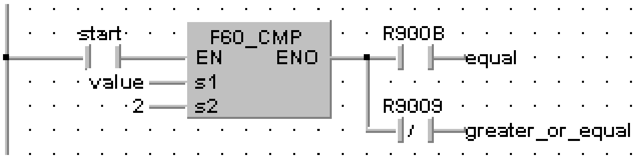


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	INT	5	
2	VAR	equal	BOOL	FALSE	set to TRUE depending on the status of R900B (= flag)
3	VAR	greater_or_equal	BOOL	FALSE	set not to TRUE depending on the status of R9009 (carry flag)

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
equal:= FALSE;
greater_or_equal:= FALSE;
IF start THEN
      F60_CMP(value, 2);
      IF R900B THEN
            equal := TRUE;
      END_IF;
      IF NOT(R9009) THEN
            greater_or_equal:= TRUE;
      END_IF;
END_IF;
```

F61_DCMP

32–bit data compare

Steps	9

Description

Compares the 32–bit data or 32–bit equivalent constant specified by **s1** with one specified by **s2** if the trigger **EN** is in the ON–state. The compare operation result is stored in special internal relays R9009, R900A to R900C.

Data	comparison between s1 and s2	Flag			
		R900A (> flag)	R900B (=flag)	R900C (< flag)	R9009 (carry–flag)
32–bit data with sign	s1< s2	OFF	OFF	ON	#
	s1=s2	OFF	ON	OFF	OFF
	s1> s2	ON	OFF	OFF	#
32–bit data without sign	s1< s2	#	OFF	#	ON
	s1=s2	OFF	ON	OFF	OFF
	s1> s2	#	OFF	#	OFF

#: turns ON or OFF depending on the conditions

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F61	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DINT, DWORD	32–bit area or 32–bit equivalent constant to be compared
s2	DINT, DWORD	32–bit area or 32–bit equivalent constant to be compared

The variables **s1** and **s2** have to be of the same data type.

Operands

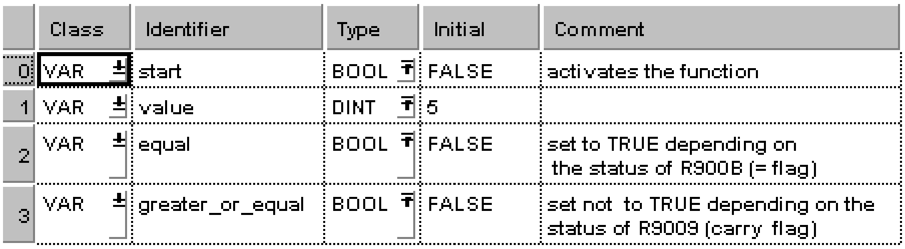
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

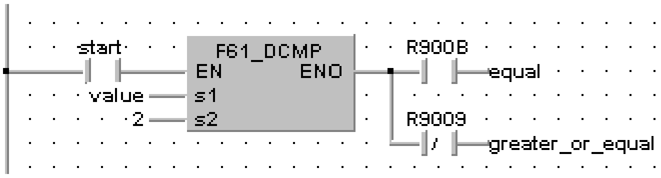


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value	DINT	5	
2	VAR	equal	BOOL	FALSE	set to TRUE depending on the status of R900B (= flag)
3	VAR	greater_or_equal	BOOL	FALSE	set not to TRUE depending on the status of R9009 (carry flag)

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
equal:= FALSE;
greater_or_equal:= FALSE;
IF start THEN
      F61_DCMP(value, 2);
      IF R900B THEN
            equal:= TRUE;
      END_IF;
      IF NOT(R9009) THEN
            greater_or_equal:= TRUE;
      END_IF;
END_IF;
```

F62_WIN

16–bit data band compare

Steps 7

Description Compares the 16–bit equivalent constant or 16–bit data specified by **s1** with the data band specified by **s2** and **s3**, if the trigger **EN** is in the ON–state. This instruction checks that **s1** is in the data band between **s2** (lower limit) and **s3** (higher limit), larger than **s3**, or smaller than **s2**. The compare operation considers +/– sign. Since the BCD data is also treated as 16–bit data with sign, we recommend using BCD data between 0 and 7999 to avoid confusion. The compare operation result is stored in special internal relays R900A, R900B, and R900C.

Comparison between s1 , s2 and s3	Flag		
	R900A (> flag)	R900B (=flag)	R900C (< flag)
s1< s2	OFF	OFF	ON
s2≤ s1≤ s3	OFF	ON	OFF
s1> s3	ON	OFF	OFF

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F62	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit area or 16–bit equivalent constant to be compared
s2	INT, WORD	lower limit, 16–bit area or 16–bit equivalent constant
s3	INT, WORD	upper limit, 16–bit area or 16–bit equivalent constant

The variables **s1, s2** and **s3** have to be of the same data type.

Operands

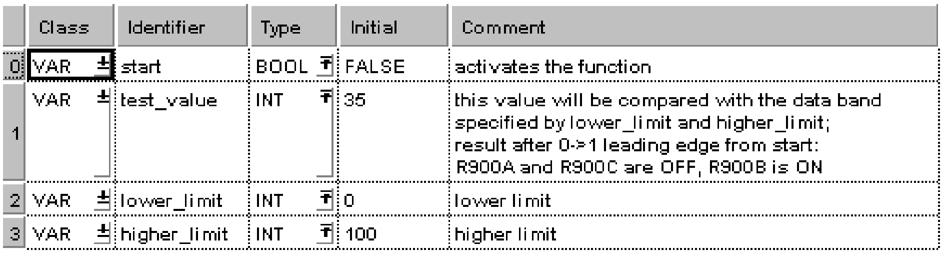
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2, s3	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

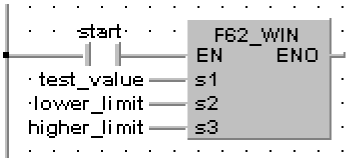


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	test_value	INT	35	this value will be compared with the data band specified by lower_limit and higher_limit; result after 0->1 leading edge from start: R900A and R900C are OFF, R900B is ON
2	VAR	lower_limit	INT	0	lower limit
3	VAR	higher_limit	INT	100	higher limit

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F62_WIN( s1_In:= test_value,
             s2_Min:= lower_limit,
             s3_Max:= higher_limit);
END_IF;
```

F63_DWIN

32–bit data band compare

Steps	13

Description Compares the 32–bit equivalent constant or 32–bit data specified by **s1** with the data band specified by **s2** and **s3**, if the trigger **EN** is in the ON–state. This instruction checks that **s1** is in the data band between **s2** (lower limit) and **s3** (higher limit), larger than **s3**, or smaller than **s2**. The compare operation considers +/– sign. Since the BCD data is also treated as 32–bit data with sign, we recommend using BCD data between 0 and 79999999 to avoid confusion. The compare operation result is stored in special internal relays R900A, R900B, and R900C.

Comparison between s1 , s2 and s3	Flag		
	R900A (> flag)	R900B (=flag)	R900C (< flag)
s1< s2	OFF	OFF	ON
s2≤ s1≤ s3	OFF	ON	OFF
s1> s3	ON	OFF	OFF

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F63	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	DINT, DWORD	32–bit area or 32–bit equivalent constant to be compared
s2	DINT, DWORD	lower limit, 32–bit area or 32–bit equivalent constant
s3	DINT, DWORD	upper limit, 32–bit area or 32–bit equivalent constant

The variables **s1, s2** and **s3** have to be of the same data type.

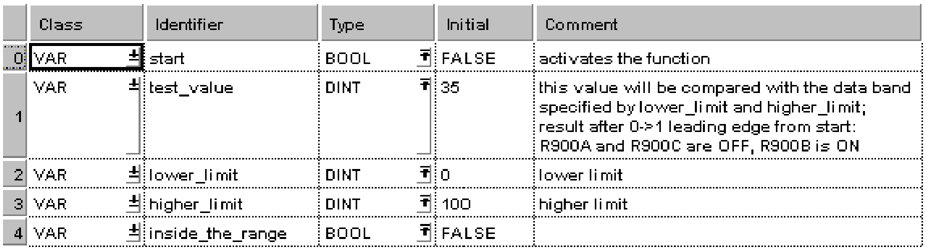
Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1, s2, s3	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

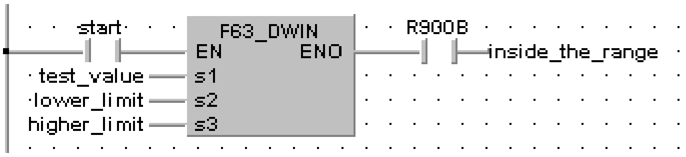
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	test_value	DINT	35	this value will be compared with the data band specified by lower_limit and higher_limit; result after 0->1 leading edge from start: R900A and R900C are OFF, R900B is ON
2	VAR	lower_limit	DINT	0	lower limit
3	VAR	higher_limit	DINT	100	higher limit
4	VAR	inside_the_range	BOOL	FALSE	

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
inside_the_range:= FALSE;
IF start THEN
      F63_DWIN( s1_In:= test_value,
             s2_Min:= lower_limit,
             s3_Max:= higher_limit);
      IF R900B THEN
            inside_the_range:= TRUE;
      END_IF;
END_IF;
```

F64_BCMP Block data compare

Steps 7

Description

Compares the contents of data block specified by **s2** with the contents of data block specified by **s3** according to the contents specified by **s1** if the trigger **EN** is in the ON–state.

s1 specifications

```
16#  1  0   0 4
     ⇑  ⇑    ⇑
     A  B    C
```

A = Starting byte position of data block specified by **s3**
1: Starting from higher order byte
0: Starting from lower lower byte
B = Starting byte position of data block specified by **s2**
1: Starting from higher order byte
0: Starting from lower order byte
C = Number of bytes to be compared
range: 16#01 to 16#99 (BCD)

The compare operation result is stored in the special internal relay R900B. When **s2** = **s3**, the special internal relay is in the ON–state.

☞ **The flag R900B used for the compare instruction is renewed each time a compare instruction is executed. Therefore the program that uses R900B should be just after F64_BCMP.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F64	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	control code specifying byte positions and number of bytes to be compared
s2	INT, WORD	starting 16–bit area to be compared to s3
s3	INT, WORD	starting 16–bit area to be compared to s2

The variables **s2** and **s3** have to be of the same data type.

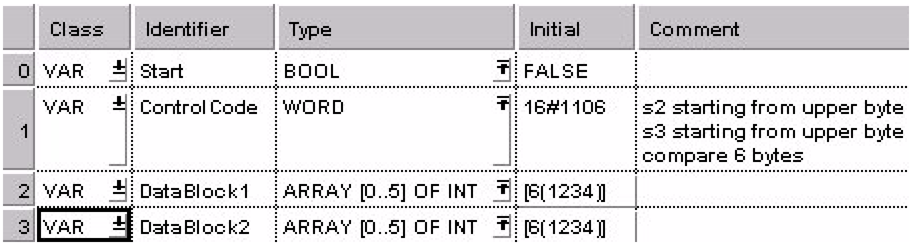
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	x
s2, s3	x	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

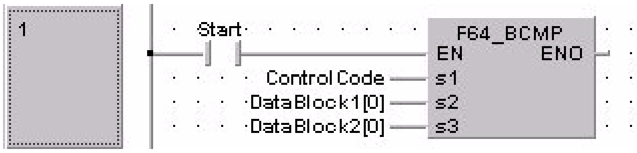
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	Control Code	WORD	16#1106	s2 starting from upper byte s3 starting from upper byte compare 6 bytes
2	VAR	DataBlock1	ARRAY [0..5] OF INT	[6(1234)]	
3	VAR	DataBlock2	ARRAY [0..5] OF INT	[6(1234)]	

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F64_BCMP( s1_Control:= ControlCode,
             s2_Start:= DataBlock1[0],
             s3_Start:= DataBlock2[0]);
END_IF;
```

Chapter 18

Logic Operation Instructions

F65_WAN

16–bit data AND

Steps	7

Description Executes AND operation of each bit in 16–bit equivalent constant or 16–bit data specified by **s1** and **s2** if the trigger **EN** is in the ON–state. The AND operation result is stored in the 16–bit area specified by **d**. When 16–bit equivalent constant is specified by **s1** or **s2**, the AND operation is performed internally converting it to 16–bit binary expression. You can use this instruction to turn OFF certain bits of the 16–bit data.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F65	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit equivalent constant or 16–bit area
s2	INT, WORD	16–bit equivalent constant or 16–bit area
d	INT, WORD	16–bit area for storing AND operation result

The variables **s1**, **s2** and **d** have to be of the same data type.

Operands

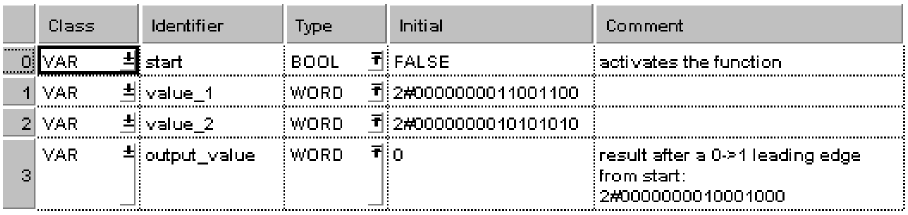
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

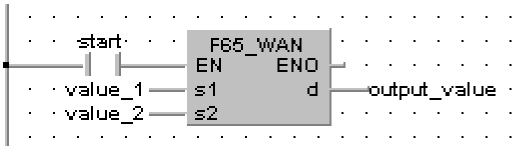


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#0000000011001100	
2	VAR	value_2	WORD	2#0000000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000010001000

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F65_WAN(value_1, value_2, output_value);
END_IF;
```

F66_WOR

16–bit data OR

Steps	7

Description

Executes OR operation of each bit in 16–bit equivalent constant or 16–bit data specified by **s1** and **s2** if the trigger **EN** is in the ON–state. The OR operation result is stored in the 16–bit area specified by **d**. When 16–bit equivalent constant is specified by **s1** or **s2**, the OR operation is performed internally converting it to 16–bit binary expression. You can use this instruction to turn ON certain bits of the 16–bit data.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F66	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit equivalent constant or 16–bit area
s2	INT, WORD	16–bit equivalent constant or 16–bit area
d	INT, WORD	16–bit area for storing OR operation result

The variables **s1**, **s2** and **d** have to be of the same data type.

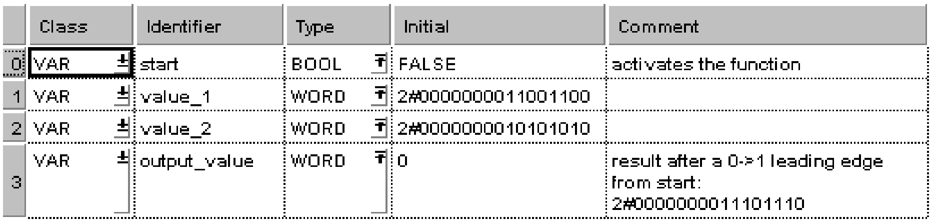
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

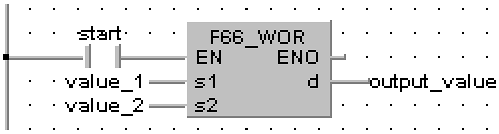
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#0000000011001100	
2	VAR	value_2	WORD	2#0000000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000011101110

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F66_WOR(value_1, value_2, output_value);
END_IF;
```

F67_XOR

16–bit data exclusive OR

Steps	7

Description Executes exclusive OR operation of each bit in 16–bit equivalent constant or 16–bit data specified by **s1** and **s2** if the trigger **EN** is in the ON–state. The exclusive OR operation result is stored in the 16–bit area specified by **d**. When 16–bit equivalent constant is specified by **s1** or **s2**, the exclusive OR operation is performed internally converting it to 16–bit binary expression.You can use this instruction to review the number of identical bits in the two 16–bit data.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F67	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit equivalent constant or 16–bit area
s2	INT, WORD	16–bit equivalent constant or 16–bit area
d	INT, WORD	16–bit area for storing XOR operation result

The variables **s1**, **s2** and **d** have to be of the same data type.

Operands

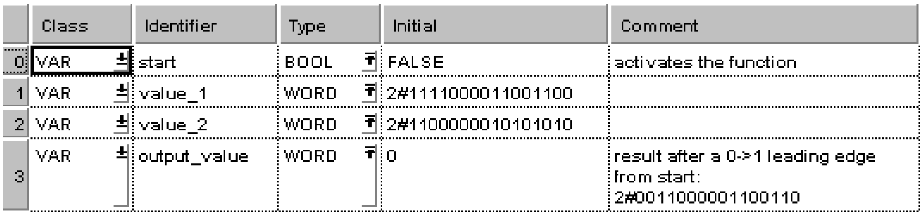
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

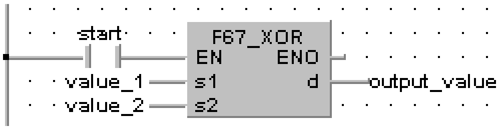


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#1111000011001100	
2	VAR	value_2	WORD	2#1100000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0011000001100110

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F67_XOR(value_1, value_2, output_value);
END_IF;
```

F68_XNR

16–bit data exclusive NOR

Steps	7

Description

Executes exclusive NOR operation of each bit in 16–bit equivalent constant or 16–bit data specified by **s1** and **s2** if the trigger **EN** is in the ON–state. The exclusive NOR operation result is stored in the 16–bit area specified by **d**. When 16–bit equivalent constant is specified by **s1** or **s2**, the exclusive NOR operation is performed internally converting it to 16–bit binary expression. You can use this instruction to review the number of identical bits in the two 16–bit data.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F68	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit equivalent constant or 16–bit area
s2	INT, WORD	16–bit equivalent constant or 16–bit area
d	INT, WORD	16–bit area for storing NOR operation result

The variables **s1**, **s2** and **d** have to be of the same data type.

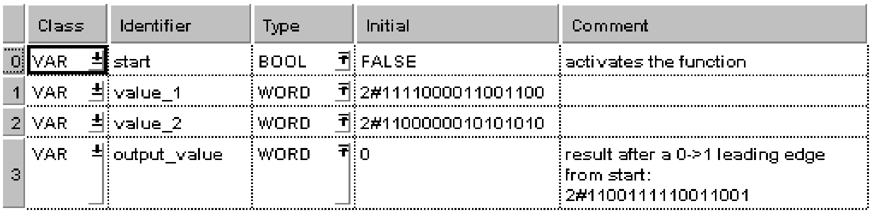
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

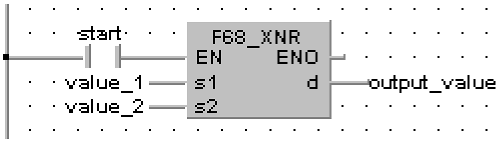
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	value_1	WORD	2#1111000011001100	
2	VAR	value_2	WORD	2#1100000010101010	
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#1100111110011001

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F68_XNR(value_1, value_2, output_value);
END_IF;
```

Chapter 19

Data Shift and Rotate Instructions

LSR

Left shift register

Steps 1

Description

Shifts 1 bit of the specified data area (WR) to the left (to the higher bit position). When programming the LSR instruction, be sure to program the data input (DataInput), shift (ShiftTrigger) and reset triggers (ResetTrigger).

DataInput specifies the state of new shift–in data: new shift–in data = 1 when the input is ON, new shift–in data = 0 when the input is OFF. **ShiftTrigger** shifts 1 bit to the left when the leading edge of the trigger is detected. **ResetTrigger** turns all the bits of the data area to 0 if the trigger is in the ON–state. The only area available for this instruction is the word internal relay (WR).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
LSR	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
DataInput	BOOL	when ON, shift–in data = 1, when OFF, shift–in data = 0
ShiftTrigger	BOOL	shifts one bit to the left when ON
ResetTrigger	BOOL	resets data area to 0 when ON
WR	INT, WORD	specified data area where data shift takes place

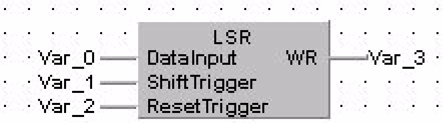
Operands

For	Relay				T/C	
	X	Y	R	L	T	C
DataInput Shift Trigger, Reset Trigger	x	x	x	x	x	x
WR	WX	WY	WR	WL	SV	EV
	–	–	x	–	–	–

x: available
–: not available

Example

Below is an example of a ladder diagram (LD) body for the instruction.



Word internal relay (WR) number range, depends on the free area in the Project –> Compile Options menu.

F100_SHR

Right shift of 16–bit data in bit units

Steps	5

Description

Shifts **n** bits of 16–bit data area specified at **d** to the right (to the lower bit position) if the trigger **EN** is in the ON–state. When **n** bits are shifted to the right, the data in the **n**th bit is transferred to special internal relay R9009 (carry–flag) and the higher **n** bits of the 16–bit data area specified by d are filled with 0s.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F100	x	x	x	x	x

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area to be shifted to the right
n	INT	number of bits to be shifted

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

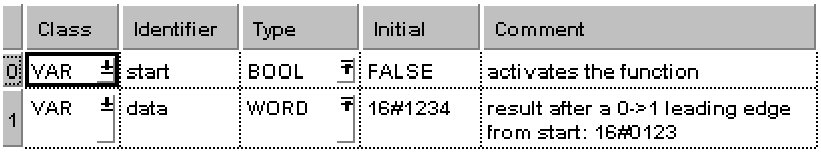
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

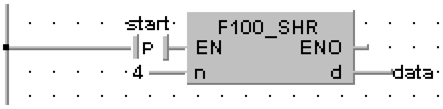


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#0123

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F100_SHR( n:= 4 ,
             d=> data );
END_IF;
```

F101_SHL

Left shift of 16–bit data in bit units

Steps	5

Description

Shifts **n** bits of 16–bit data area specified at **d** to the left (to the higher bit position) if the trigger **EN** is in the ON–state. When **n** bits are shifted to the left, the data in the **n**th bit is transferred to special internal relay R9009 (carry–flag) and **n** bits starting with bit position 0 are filled with 0s.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F101	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area to be shifted to the left
n	INT	number of bits to be shifted

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

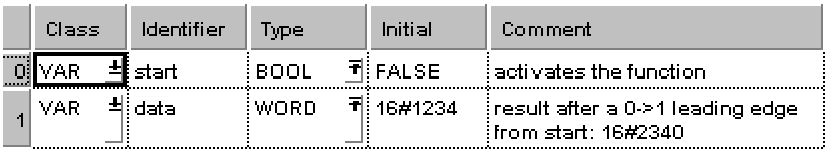
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

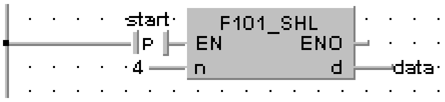


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F101_SHL( n:= 4,
             d=> data);
END_IF;
```

F105_BSR

Right shift of one hexadecimal digit (4 bits) of 16–bit data

Steps 3

Description Shifts one hexadecimal digit (4 bits) of the 16–bit area specified by **d** to the right (to the lower digit position) if the trigger **EN** is in the ON–state. When one hexadecimal digit (4 bits) is shifted to the right,

- hexadecimal digit position 0 (bit position 0 to 3) of the data specified by **d** is shifted out and is transferred to the lower digit (bit position 0 to 3) of special data register DT9014) and
- hexadecimal digit position 3 (bit position 12 to 15) of the 16–bit area specified by **d** becomes 0.

This instruction is useful when the hexadecimal or BCD data is treated.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F105	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area to be shifted to the right

Operands

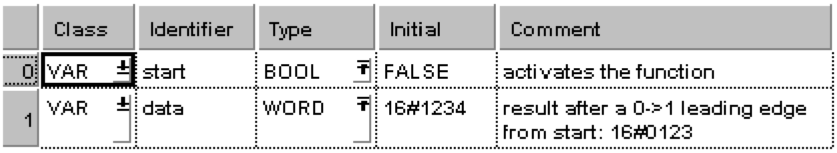
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

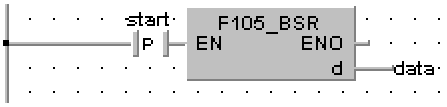


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#0123

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F105_BSR(data);
END_IF;
```

F106_BSL

Left shift of one hexadecimal digit (4 bits) of 16–bit data

Steps	3

Description Shifts one hexadecimal digit (4 bits) of the 16–bit area specified by **d** to the left (to the higher digit position) if the trigger **EN** is in the ON–state. When one hexadecimal digit (4 bits) is shifted to the left,

- hexadecimal digit position 3 (bit position 12 to 15) of the data specified by **d** is shifted out and is transferred to the lower digit (bit position 0 to 3) of special data register DT9014 (DT90014 for FP10/10S).
- hexadecimal digit position 0 (bit position 0 to 3) of the 16–bit area specified by **d** becomes 0.

This instruction is useful when the hexadecimal or BCD data is treated.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F106	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area to be shifted to the left

Operands

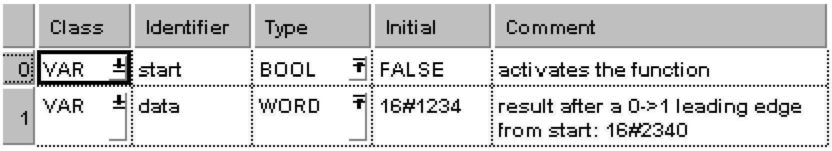
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

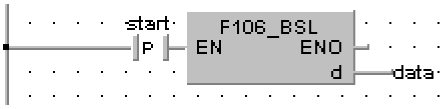


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	data	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F106_BSL(data);
END_IF;
```

F110_WSHR Right shift of one word (16 bits) of 16–bit data range

Steps	5

Description Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower word address) if the trigger **EN** is in the ON–state. When one word (16 bits) is shifted to the right,

- the starting word is shifted out
- the data in the ending word becomes 0

d1 and **d2** should be:

- in the same operand
- **d1** ≤ **d2**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F110	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16–bit area
d2	INT, WORD	ending 16–bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

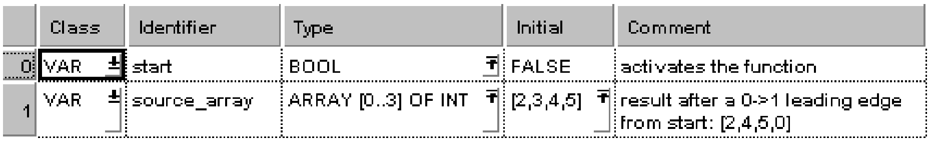
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1, d2	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

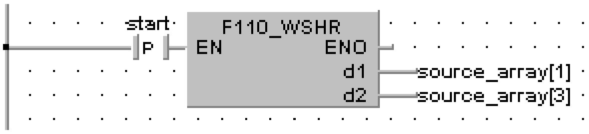


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF INT	[2,3,4,5]	result after a 0->1 leading edge from start: [2,4,5,0]

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F110_WSHR( d1_Start=> source_array[1],
             d2_End=> source_array[3]);
END_IF;
```

F111_WSHL Left shift of one word (16 bits) of 16–bit data range

Steps	5

Description Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher word address) if the trigger **EN** is in the ON–state. When one word (16 bits) is shifted to the left,

- the ending word is shifted out
- the data in the starting word becomes 0

d1 and **d2** should be:

- in the same operand
- **d1** ≤ **d2**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F111	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16–bit area
d2	INT, WORD	ending 16–bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

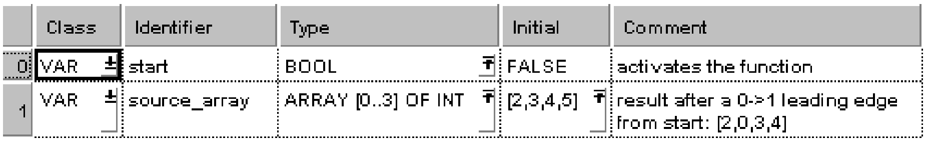
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1, d2	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

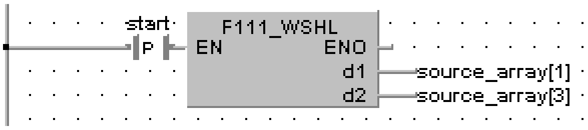


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF INT	[2,3,4,5]	result after a 0->1 leading edge from start: [2,0,3,4]

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F111_WSHL( d1_Start=> source_array[1],
              d2_End=> source_array[3]);
END_IF;
```

F112_WBSR Right shift of one hex. digit (4 bits) of 16–bit data range

Steps	5

Description Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower digit position) if the trigger **EN** is in the ON–state. When one hexadecimal digit (4 bits) is shifted to the right,

- the data in the lower hexadecimal digit (bit position 0 to 3) of the 16–bit data specified by **d1** is shifted out
- the data in the higher hexadecimal digit (bit position 12 to 15) of the 16–bit data specified by **d2** becomes 0

d1 and **d2** should be:

- in the same operand
- **d1** ≤ **d2**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F112	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16–bit area
d2	INT, WORD	ending 16–bit area

The variables **d1** and **d2** have to be of the same data type.

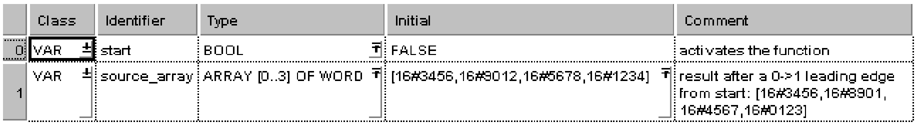
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1, d2	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

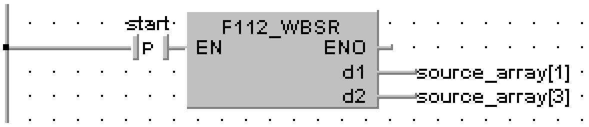
POU header
In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF WORD	[16#3456,16#9012,16#5678,16#1234]	result after a 0->1 leading edge from start: [16#3456,16#8901, 16#4567,16#0123]

Body
When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F112_WBSR( d1_Start=> source_array[1],
             d2_End=> source_array[3]);
END_IF;
```

F113_WBSL

Left shift of one hex. digit (4 bits) of 16–bit data range

Steps	5

Description

Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher digit position) if the trigger **EN** is in the ON–state. When one hexadecimal digit (4 bits) is shifted to the left,

- the data in the higher hexadecimal digit (bit position 12 to 15) of the 16–bit data specified by **d2** is shifted out.
- the data in the lower hexadecimal digit (bit position 0 to 3) of the 16–bit data specified by **d1** becomes 0.

d1 and **d2** should be:

- in the same operand
- **d1** ≤ **d2**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F113	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d1	INT, WORD	starting 16–bit area
d2	INT, WORD	ending 16–bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

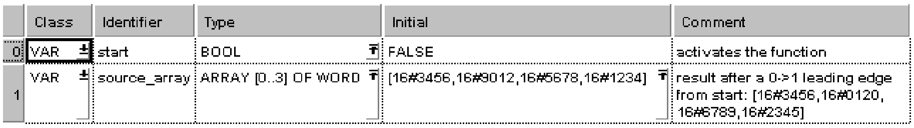
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1, d2	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

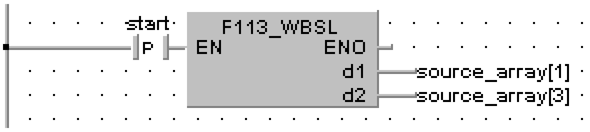


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	source_array	ARRAY [0..3] OF WORD	[16#3456,16#9012,16#5678,16#1234]	result after a 0->1 leading edge from start: [16#3456,16#0120, 16#6789,16#2345]

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
        F113_WBSL( d1_Start=> source_array[1],
                 d2_End=> source_array[3]);
END_IF;
```

F119_LRSR

LEFT/RIGHT shift register

Steps 5

Description

LR_trig: Left/right trigger; specifies the direction of the shift–out. **LR_trig** = ON: shifts out to the left, **LR_trig = OFF**: shifts out to the right.

DataInp: Specifies the new shift–in data. New shift–in data = 1 when the data input is in the ON–state. New shift–in data = 0 when the data input is in the OFF–state.

Sh_trig: Shifts 1 bit to the left or right when the leading edge of the trigger is detected (OFF → ON).

Rst_trig: Turns all the bits of the data range specified by d1 and d2 to 0 if this trigger is in the ON–state.

d1: Start of 16 bit area.

d2: End of 16 bit area.

Carry: Shifted–out bit.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F119	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
LR_trig	BOOL	specifies direction of shift, ON = left, OFF = right
DataInp	BOOL	shift–in data, ON = 1, OFF = 0
Sh_trig	BOOL	activates shift
Rst_trig	BOOL	resets data in area specified by d1 and d2 to 0
Carry	BOOL	bit shifted out
d1	INT, WORD	starting 16–bit area
d2	INT, WORD	ending 16–bit area

The variables **d1** and **d2** have to be of the same data type.

Operands

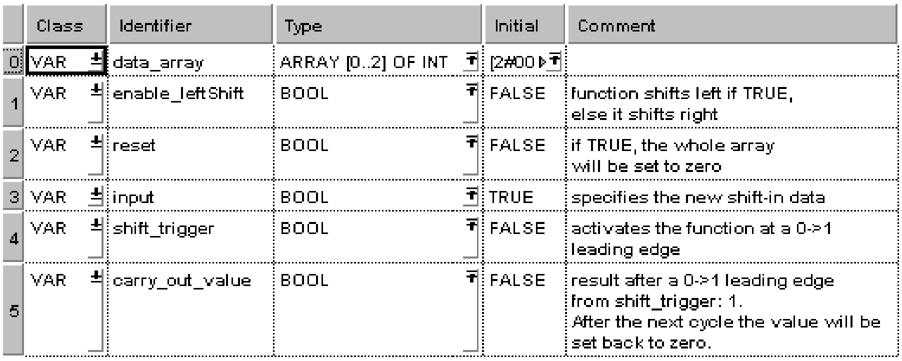
For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
LR_trig, DataInp, Sh_trig, Rst_trig	x	x	x	x	x	x	–	–	–	–
Carry	–	x	x	x	x	x	–	–	–	–
d1, d2	**WX**	**WY**	**WR**	**WL**	**SV**	**EV**	**DT**	**LD**	**FL**	**dec. or hex.**
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

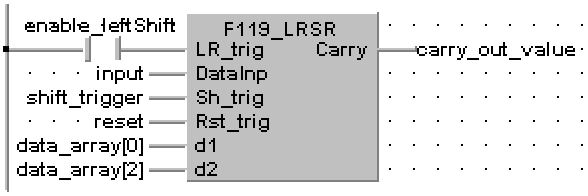


	Class	Identifier	Type	Initial	Comment
0	VAR	data_array	ARRAY [0..2] OF INT	[2#00	
1	VAR	enable_leftShift	BOOL	FALSE	function shifts left if TRUE, else it shifts right
2	VAR	reset	BOOL	FALSE	if TRUE, the whole array will be set to zero
3	VAR	input	BOOL	TRUE	specifies the new shift-in data
4	VAR	shift_trigger	BOOL	FALSE	activates the function at a 0->1 leading edge
5	VAR	carry_out_value	BOOL	FALSE	result after a 0->1 leading edge from shift_trigger: 1. After the next cycle the value will be set back to zero.

Body

When the variable *enable_leftShift* is set to TRUE, the function shifts left, else it shifts right.

LD



ST

```
carry_out_value:=F119_LRSR( LR_trig:= enable_leftShift,
                 DataInp:= input,
                 Sh_trig:= shift_trigger,
                 Rst_trig:= reset,
                 d1:= data_array[0],
                 d2:= data_array[2]);
```

F120_ROR

16–bit data right rotate

Steps	5

Description

Rotates **n** bits of the 16–bit data specified by **d** to the right if the trigger **EN** is in the ON–state. When **n** bits are rotated to the right,

- the data in bit position **n**–1 (**n**th bit starting from bit position 0) is transferred to the special internal relay R9009 (carry–flag)
- **n** bits starting from bit position 0 are shifted out to the right and into the higher bit positions of the 16–bit data specified by **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F120	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	number of bits to be rotated

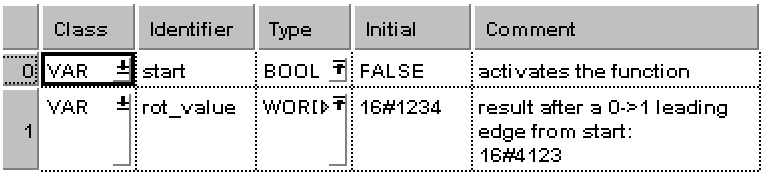
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

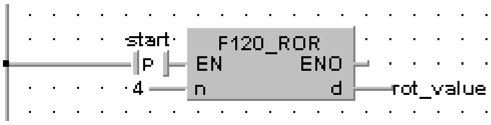
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#4123

Body When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F120_ROR( n:= 4,
             d=> rot_value);
END_IF;
```

F121_ROL

16–bit data left rotate

Steps	5

Description Rotates **n** bits of the 16–bit data specified by **d** to the left if the trigger **EN** is in the ON–state. When **n** bits are rotated to the left,

- the data in bit position 16–**n** (**n**th bit starting from bit position 15) is transferred to special internal relay R9009 (carry–flag)
- **n** bits starting from bit position 15 are shifted out to the left and into the lower bit positions of the 16–bit data specified by **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F121	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	number of bits to be rotated

Operands

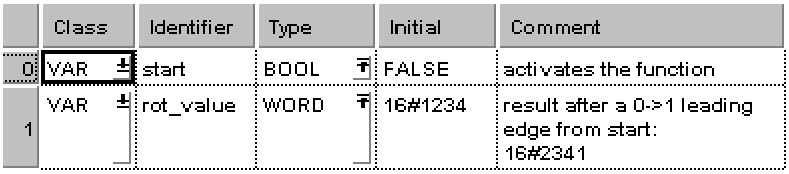
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

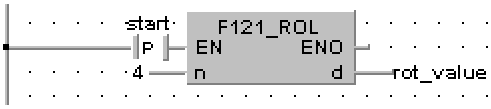


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#2341

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
        F121_ROL( n:= 4,
                 d=> rot_value);
END_IF;
```

F122_RCR

16–bit data right rotate with carry–flag data

Steps	5

Description

Rotates **n** bits of the 16–bit data specified by **d** including the data of carry–flag to the right if the trigger **EN** is in the ON–state. When **n** bits with carry–flag data are rotated to the right,

- the data in bit position **n**–1 (**n**th bit starting from bit position 0) are transferred to special internal relay R9009 (carry–flag)
- **n** bits starting from bit position 0 are shifted out to the right and carry–flag data and **n**–1 bits starting from bit position 0 are subsequently shifted into the higher bit positions of the 16–bit data specified by **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F122	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	number of bits to be rotated

Operands

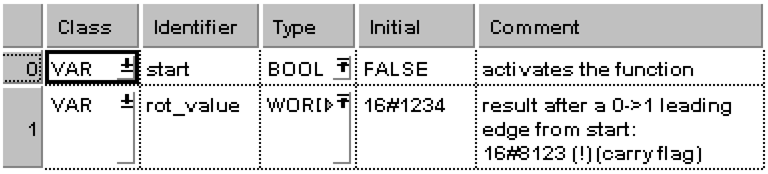
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

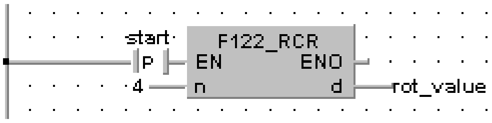


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#8123 (!) (carry flag)

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F122_RCR( n:= 4,
             d=> rot_value);
END_IF;
```

F123_RCL 16–bit data left rotate with carry–flag data

Steps	5

Description Rotates **n** bits of the 16–bit data specified by **d** including the data of carry–flag to the left if the trigger **EN** is in the ON–state. When **n** bits with carry–flag data are rotated to the left,

- the data in bit position 16–**n** (**n**th bit starting from bit position 15) is transferred to special internal relay R9009 (carry–flag).
- **n** bits starting from bit position 15 are shifted out to the left and carry–flag data and **n**–1 bits starting from bit position 15 are shifted into lower bit positions of the 16–bit data specified by **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F123	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	number of bits to be rotated

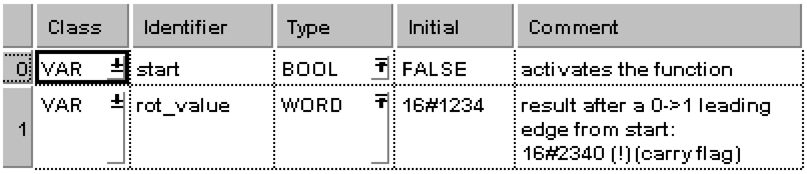
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

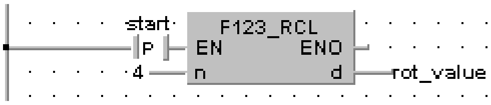
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	rot_value	WORD	16#1234	result after a 0->1 leading edge from start: 16#2340 (!) (carry flag)

Body When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F123_RCL( n:= 4,
             d=> rot_value);
END_IF;
```

Chapter 20

Data Conversion Instructions

F70_BCC

Block check code calculation

Steps	9

Description

Calculates the Block Check Code (BCC), which is used to detect errors in message transmissions, of **s3** bytes of ASCII data starting from the 16-bit area specified by **s2** according to the calculation method specified by **s1**. The Block Check Code (BCC) is stored in the lower byte of the 16-bit area specified by **d**. (BCC is one byte. The higher byte of **d** does not change.)

s1: Specifying the Block Check Code (BCC) calculation method:

- 0: Addition
- 1: Subtraction
- 2: Exclusive OR operation

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F70	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT	specifies BCC calculation method: 0 = addition, 1 = subtraction, 2 = exclusive OR operation
s2	WORD, INT	starting 16–bit area to calculate BCC
s3	INT	specifies number of bytes for BCC calculation
d	WORD, INT	16–bit area for storing BCC

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s3	x	x	x	x	x	x	x	x	x	x
s2	x	x	x	x	x	x	x	x	x	–
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

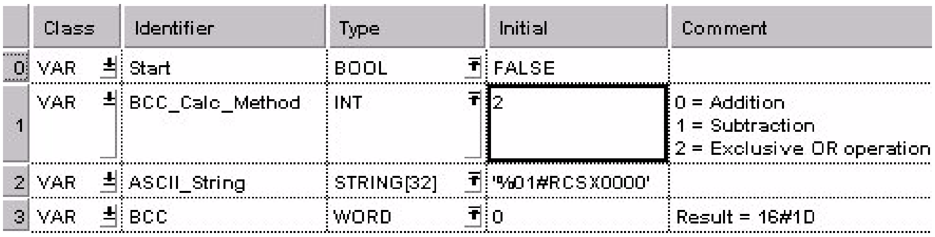
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the number of specified bytes for the target data exceeds the limit of the specified data area.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	BCC_Calc_Method	INT	2	0 = Addition 1 = Subtraction 2 = Exclusive OR operation
2	VAR	ASCII_String	STRING[32]	'%01#RCSX0000'	
3	VAR	BCC	WORD	0	Result = 16#1D

Body

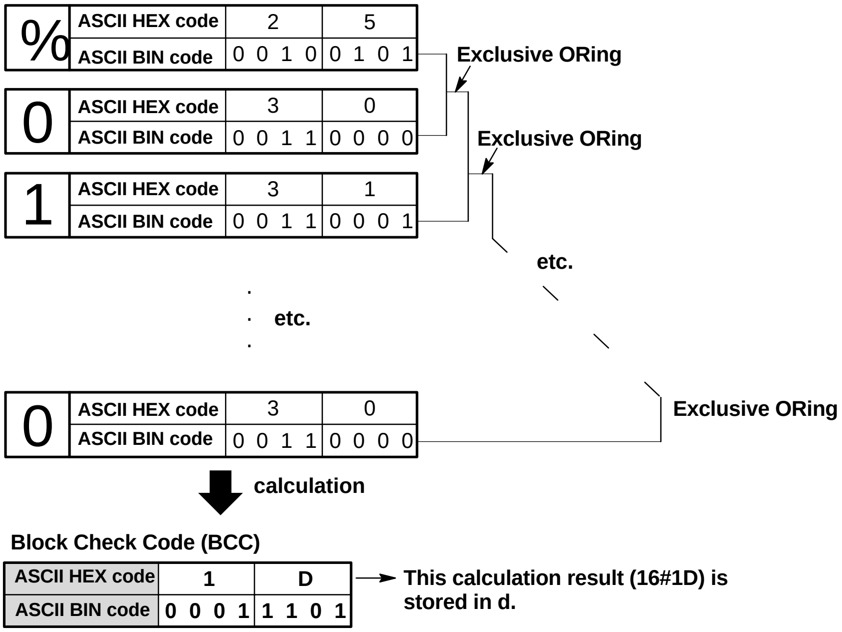
A block check code is performed on the value entered for the variable *ASCII_String* when *Start* becomes TRUE. The exclusive OR operation, which is more suitable when large amounts of data are transmitted, has been chosen for the BCC method.

How the BCC is calculated using the exclusive OR operation:

Exclusive OR operation:

In1	In2	Out
0	0	0
0	1	1
1	0	1
1	1	0

XOR
In1
In2
Out



%	ASCII HEX code	2	5
	ASCII BIN code	0 0 1 0	0 1 0 1

0	ASCII HEX code	3	0
	ASCII BIN code	0 0 1 1	0 0 0 0

1	ASCII HEX code	3	1
	ASCII BIN code	0 0 1 1	0 0 0 1

0	ASCII HEX code	3	0
	ASCII BIN code	0 0 1 1	0 0 0 0

Block Check Code (BCC)

ASCII HEX code	1	D
ASCII BIN code	0 0 0 1	1 1 0 1

This calculation result (16#1D) is stored in d.

The ASCII BIN code bits of the first two characters are compared with each other to yield an 8–character exclusive OR operation result:

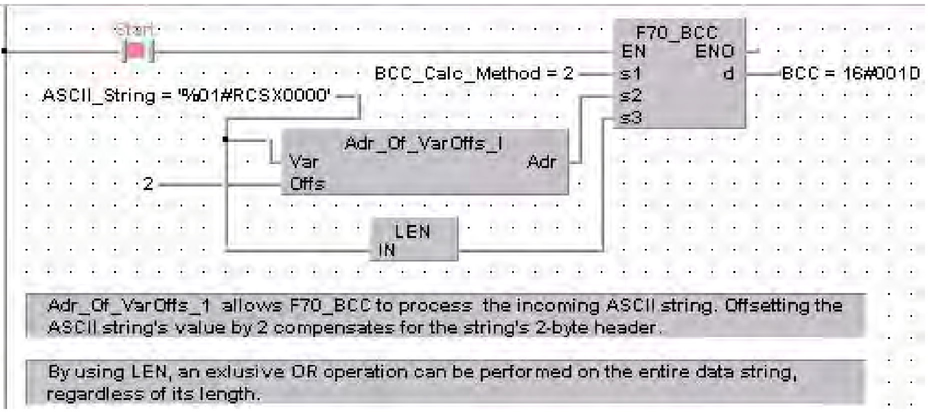
Sign for comparison	ASCII BIN code
%	00100101
0	00110000
Exclusive OR result	00010101

This result is then compared to the ASCII BIN code of the next character, i.e. “1”.

Sign for comparison	ASCII BIN code
Exclusive OR result	00010101
1	00110001
Next exclusive OR	00100100

And so on until the final character is reached.

LD



Adr_Of_VarOffs_1 allows F70_BCC to process the incoming ASCII string. Offsetting the ASCII string's value by 2 compensates for the string's 2-byte header.

By using LEN, an exlusive OR operation can be performed on the entire data string, regardless of its length.

ST

```
IF start THEN
      F70_BCC( s1_Control:= BCC_Calc_Methode,
                  s2_Start:=     Adr_Of_VarOffs(     Var:=
ASCII_String,
             Offs:= 2),
             s3_Number:= LEN( ASCII_String),
             d=> BCC);
END_IF;
```

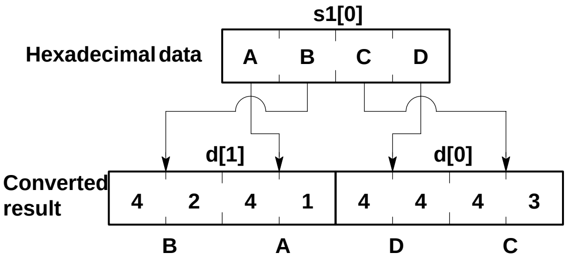
F71_HEX2A

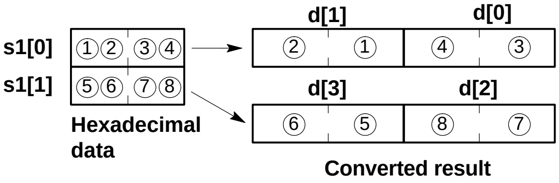
HEX → ASCII conversion

Steps	7

Description Converts the data of **s2** bytes starting from the 16–bit area specified by **s1** to ASCII codes that express the equivalent hexadecimals if the trigger **EN** is in the ON–state. The number of bytes to be converted is specified by **s2**. The converted result is stored in the area starting with the 16–bit area specified by **d**. ASCII code requires 8 bits (one byte) to express one hexadecimal character. Upon conversion to ASCII, the data length will thus be twice the length of the source data.

The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data.





ASCII HEX codes to express hexadecimal characters:

Hexadecimal number	ASCII HEX code
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39
A	16#41
B	16#42
C	16#43
D	16#44
E	16#45
F	16#46

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F71	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	starting 16–bit area for hexadecimal number (source)
s2	INT	specifies number of source data bytes to be converted
d	WORD	starting 16–bit area for storing ASCII code (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

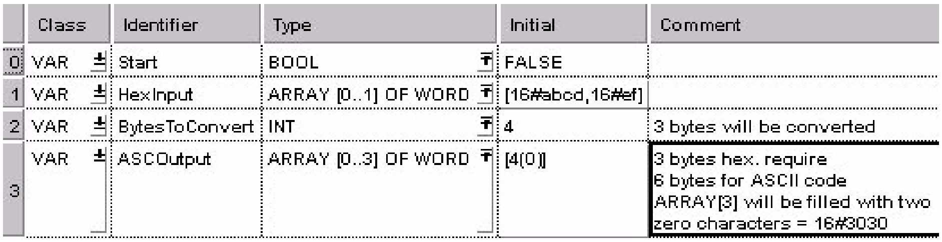
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the byte number specified by s2 exceeds the area specified by s1 – the calculated result exceeds the area specified by d. – the data specified by s2 is recognized as “0”.
R9008	%MX0.900.8	for an instant	

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

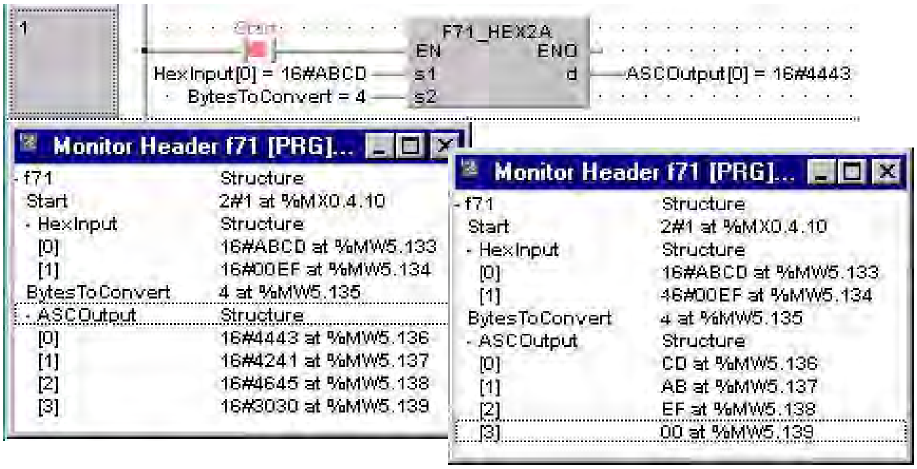


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	HexInput	ARRAY [0..1] OF WORD	[16#abcd,16#ef]	
2	VAR	BytesToConvert	INT	4	3 bytes will be converted
3	VAR	ASCOutput	ARRAY [0..3] OF WORD	[4(0)]	3 bytes hex. require 6 bytes for ASCII code ARRAY[3] will be filled with two zero characters = 16#3030

Body

When the variable *Start* is set to true, the number of data bytes given in *BytesToConvert* in *HexInput* is converted to ASCII code and stored in *ASCIIOutput*. Note that two characters that make up one byte are interchanged when stored. One Monitor Header shows the Hex values, and the other the ASCII values.

LD



```
Monitor Header f71 [PRG]...
- f71              Structure
  Start            2#1 at %MX0.4.10
  - HexInput       Structure
    [0]            16#ABCD at %MW5.133
    [1]            16#00EF at %MW5.134
  BytesToConvert   4 at %MW5.135
  - ASCOutput      Structure
    [0]            16#4443 at %MW5.136
    [1]            16#4241 at %MW5.137
    [2]            16#4645 at %MW5.138
    [3]            16#3030 at %MW5.139
```

```
Monitor Header f71 [PRG]...
- f71              Structure
  Start            2#1 at %MX0.4.10
  - HexInput       Structure
    [0]            16#ABCD at %MW5.133
    [1]            46#00EF at %MW5.134
  BytesToConvert   4 at %MW5.135
  - ASCOutput      Structure
    [0]            CD at %MW5.136
    [1]            AB at %MW5.137
    [2]            EF at %MW5.138
    [3]            00 at %MW5.139
```

ST

```
IF start THEN
      F71_HEX2A( s1_Start:= HexInput[0],
             s2_Number:= BytesToConvert,
             d_Start=> ASCOutput[0]);
END_IF;
```

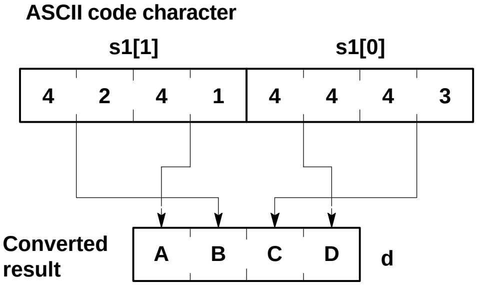
F72_A2HEX

ASCII → HEX conversion

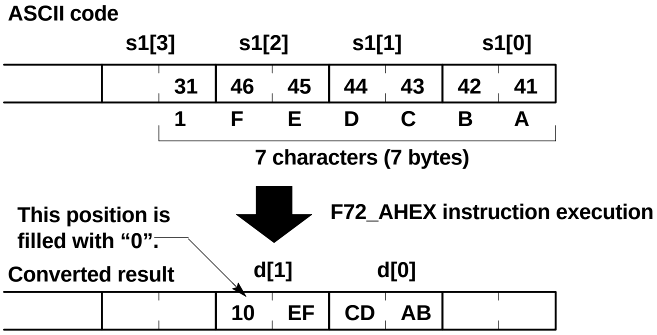
Steps	7

Description Converts the ASCII codes that express the hexadecimal characters starting from the 16–bit area specified by **s1** to hexadecimal numbers if the trigger **EN** is in the ON–state. **s2** specifies the number of ASCII (number of characters) to be converted. The converted result is stored in the area starting from the 16–bit area specified by **d**. ASCII code requires 8 bits (one byte) to express one hexadecimal character. Upon conversion to a hexadecimal number, the data length will thus be half the length of the ASCII code source data.

The data for two ASCII code characters is converted to two numeric digits for one word. When this takes place, the characters of the upper and lower bytes are interchanged. Four characters are converted as one segment of data.



Converted results are stored in byte units. If an odd number of characters is being converted, “0” will be entered for bits 0 to 3 of the final data (byte) of the converted results. Conversion of odd number of source data bytes:



Hexadecimal characters and ASCII codes:

ASCII HEX code	Hexadecimal number
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9
16#41	A
16#42	B
16#43	C
16#44	D
16#45	E
16#46	F

Data types

Variable	Data type	Function
s1	WORD	starting 16–bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	INT, WORD	starting 16–bit area for storing converted data (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

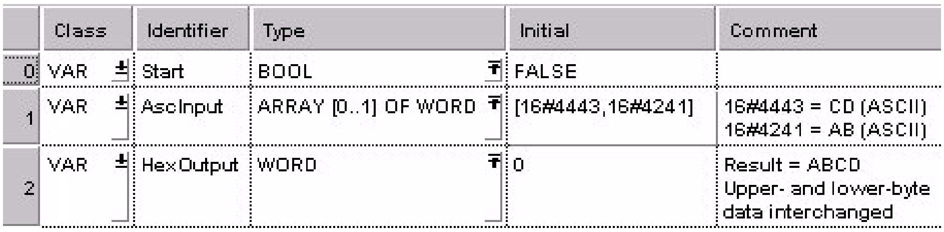
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the number of bytes specified by s2 exceeds the area specified by s1. – the converted result exceeds the area specified by d. – the data specified by s2 is recognized as “0”. – ASCII code, not a hexadecimal number (0 to F), is specified.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

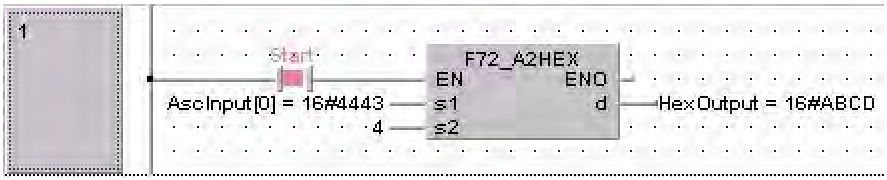


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	AscInput	ARRAY [0..1] OF WORD	[16#4443,16#4241]	16#4443 = CD (ASCII) 16#4241 = AB (ASCII)
2	VAR	HexOutput	WORD	0	Result = ABCD Upper- and lower-byte data interchanged

Body

When the variable *Start* is set to TRUE, the function is executed. In this example, the value for s2, i.e. the number of bytes to be converted from ASCII code to hexadecimal code, is entered directly at the contact pin.

LD



ST

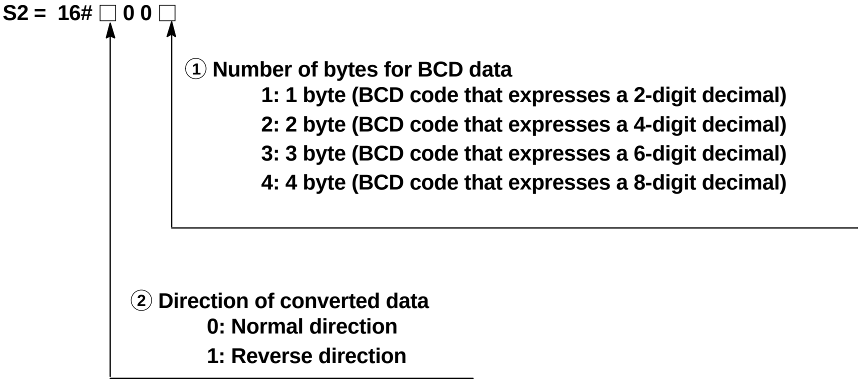
```
IF start THEN
       F72_A2HEX( s1_Start:= AscInput[0],
               s2_Number:= 4,
               d_Start=> HexOutput);
```

F73_BCD2A

BCD → ASCII conversion

Steps	7

Description Converts the BCD code starting from the 16–bit area specified by **s1** to the ASCII code that expresses the equivalent decimals according to the contents specified by **s2** if the trigger **EN** is in the ON–state. **s2** specifies the number of source data bytes and the direction of converted data (normal/reverse).



S2 = 16# □ 0 0 □

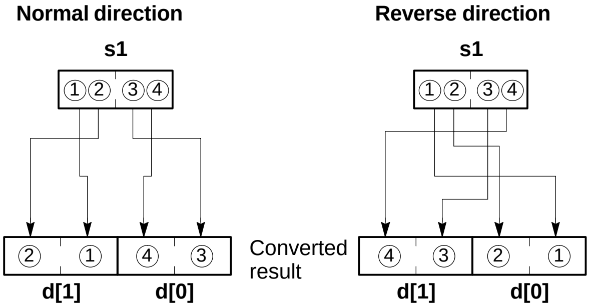
① Number of bytes for BCD data

1: 1 byte (BCD code that expresses a 2-digit decimal)
2: 2 byte (BCD code that expresses a 4-digit decimal)
3: 3 byte (BCD code that expresses a 6-digit decimal)
4: 4 byte (BCD code that expresses a 8-digit decimal)

② Direction of converted data

0: Normal direction
1: Reverse direction

The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data:



The converted result is stored in the area specified by **d**. ASCII code requires 8 bits (one byte) to express one BCD character. Upon conversion to ASCII, the data length will thus be twice the length of the BCD source data.

ASCII HEX code to express BCD character:

BCD character	ASCII HEX code
0	H30
1	H31
2	H32
3	H33
4	H34
5	H35
6	H36
7	H37
8	H38
9	H39

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F73	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	starting 16–bit area for BCD data (source)
s2	INT, WORD	specifies number of source data bytes to be converted, and how it is arranged
d	WORD	starting 16–bit area for storing converted result (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

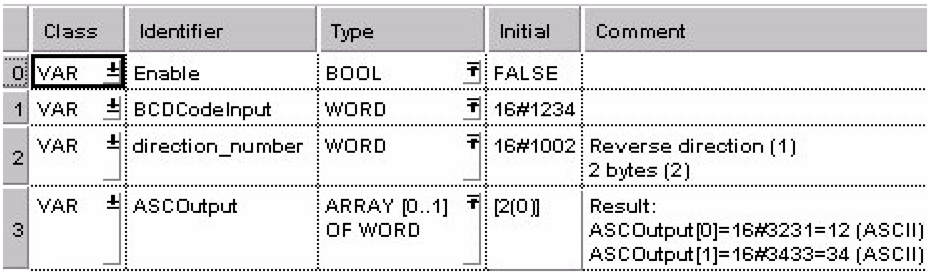
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the data specified by s1 is not BCD data. – the number of bytes specified by s2 exceeds the area specified by s1. – the converted result exceeds the area specified by d. – the data specified by s2 is recognized as “0”. – the number of bytes specified by s2 is more than 16#4.
R9008	%MX0.900.8	for an instant	

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

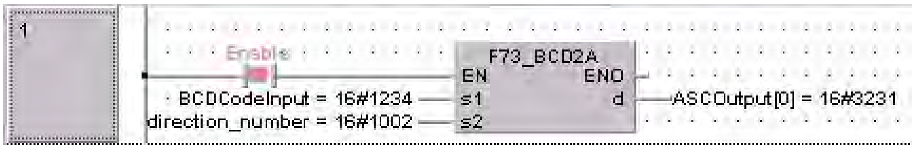


	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	BCDCodeInput	WORD	16#1234	
2	VAR	direction_number	WORD	16#1002	Reverse direction (1) 2 bytes (2)
3	VAR	ASCOutput	ARRAY [0..1] OF WORD	[2(0)]	Result: ASCOutput[0]=16#3231=12 (ASCII) ASCOutput[1]=16#3433=34 (ASCII)

Body

When the global variable *Enable* is set to TRUE, the function is executed. In this example, the variable *direction_number* specifies that from the input variable *BCDCodeInput*, 2 bytes will be converted in the reverse direction and stored in *ASCIIOutput*.

LD



ST

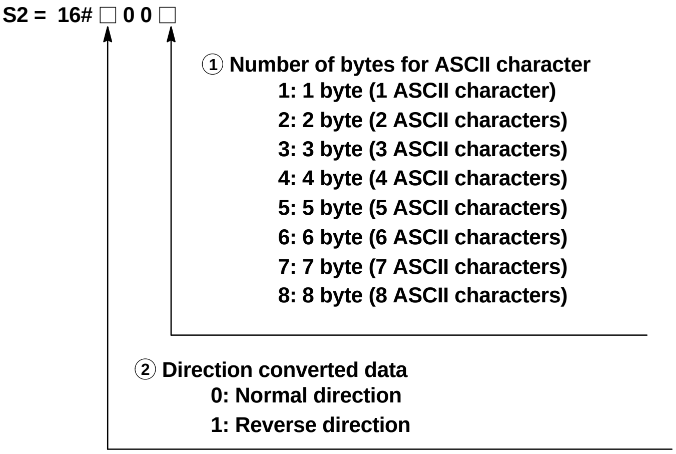
```
IF start THEN
      F73_BCD2A( s1_Start:= BCDCodeInput ,
             s2_Number:= direction_number ,
             d_Start=> ASCOutput[0] );
END_IF;
```

F74_A2BCD

ASCII → BCD conversion

Steps	9

Description Converts the ASCII codes that express the decimal characters starting from the 16–bit area specified by **s1** to BCD if the trigger **EN** is in the ON–state. **s2** specifies the number of source data bytes and the direction of converted code source data.

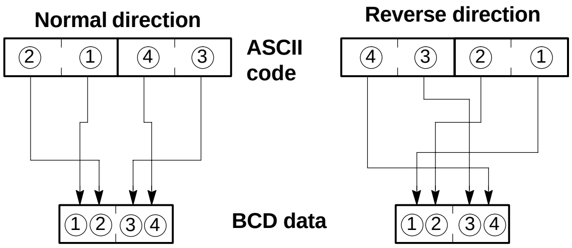


S2 = 16# □ 0 0 □

① Number of bytes for ASCII character
1: 1 byte (1 ASCII character)
2: 2 byte (2 ASCII characters)
3: 3 byte (3 ASCII characters)
4: 4 byte (4 ASCII characters)
5: 5 byte (5 ASCII characters)
6: 6 byte (6 ASCII characters)
7: 7 byte (7 ASCII characters)
8: 8 byte (8 ASCII characters)

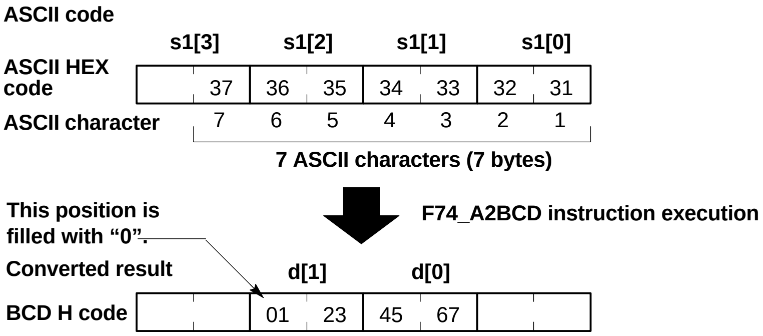
② Direction converted data
0: Normal direction
1: Reverse direction

Four characters are converted as one segment of data:



The converted result is stored in byte units in the area starting from the 16–bit area specified by **d**. ASCII code requires 8 bits (1 byte) to express 1 BCD character. Upon conversion to a BCD number, the data length will thus be half the length of the ASCII code source data.

If an odd number of characters is being converted, “0” will be entered for bit position 0 to 3 of the final data (byte) of the converted results if data is sequenced in the normal direction, and “0” will be entered for bit position 4 to 7 if data is being sequenced in the reverse direction:



ASCII code

	s1[3]		s1[2]		s1[1]		s1[0]	
ASCII HEX code		37	36	35	34	33	32	31
ASCII character		7	6	5	4	3	2	1

7 ASCII characters (7 bytes)

F74_A2BCD instruction execution

This position is filled with “0”.

Converted result

			d[1]		d[0]			
BCD H code			01	23	45	67		

ASCII HEX code to express BCD character:

BCD character	ASCII HEX code
0	H30
1	H31
2	H32
3	H33
4	H34
5	H35
6	H36
7	H37
8	H38
9	H39

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F74	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	starting 16–bit area for storing ASCII code (source)s
s2	INT, WORD	specifies number of source data bytes to be converted, and how it is arranged
d	WORD	starting 16–bit area for storing converted result (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

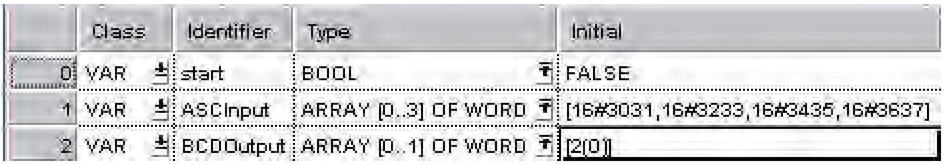
x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– ASCII code not corresponding to decimal numbers (0 to 9) is specified. – the number of bytes specified by s2 exceeds the area specified by s1. – the converted result exceeds the area specified by d. – the data specified by s2 is recognized as “0”. – the number of bytes for ASCII characters in s2 is more than 16#8.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

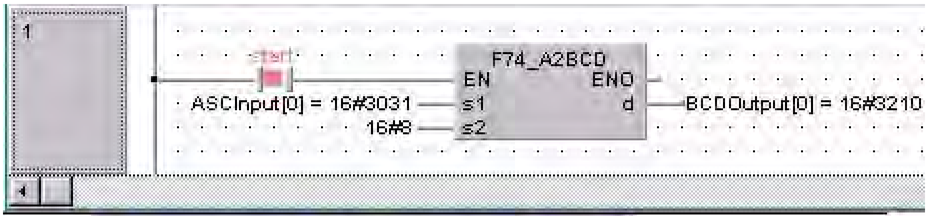
POU header In the POU header, all input and output variables are declared that are used for programming this function.

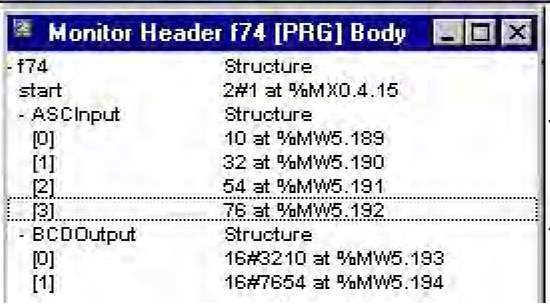


	Class	Identifier	Type	Initial
0	VAR	start	BOOL	FALSE
1	VAR	ASCInput	ARRAY [0..3] OF WORD	[16#3031,16#3233,16#3435,16#3637]
2	VAR	BCDOutput	ARRAY [0..1] OF WORD	[2(0)]

Body When the variable *start* is set to TRUE, the function is executed. For the variable at s1, you never need define an ARRAY with more than four elements because 8 ASCII characters require 8 bytes of memory and the function cannot convert more than 8 bytes. In this example, the value for s2 is entered directly at the contact pin.

LD





```
- f74             Structure
  start           2#1 at %MX0.4.15
  - ASCInput      Structure
    [0]           10 at %MW5.189
    [1]           32 at %MW5.190
    [2]           54 at %MW5.191
    [3]           76 at %MW5.192
  - BCDOutput     Structure
    [0]           16#3210 at %MW5.193
    [1]           16#7654 at %MW5.194
```

Set to display ASCII characters

ST

```
IF start THEN
       F74_A2BCD( s1_Start:= ASCInput[0] ,
              s2_Number:= 16#8 ,
              d_Start=> BCDOutput[0] );
END_IF;
```

F75_BIN2A

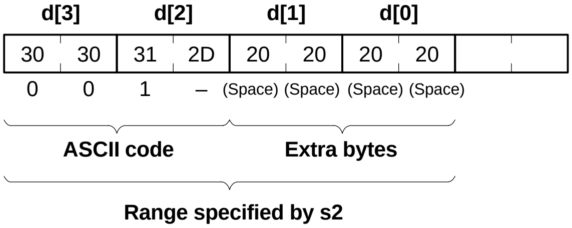
16–bit BIN → ASCII conversion

Steps **7**

Description Converts the 16-bit data specified by **s1** to ASCII codes that express the equivalent decimal value. The converted result is stored in the area starting from the 16-bit area specified by **d** as specified by **s2**. Specify the number of bytes in decimal number in **s2**. (This specification cannot be made with BCD data.)

- If a positive number is converted, the “+” sign is not converted.
- When a negative number is converted, the “–” sign is also converted to ASCII code (ASCII HEX code: 16#2D).
- If the area specified by **s2** is more than that required by the converted data the ASCII code for “SPACE” (ASCII HEX code: 16#20) is stored in the extra area.
- Data is stored in the direction towards the final address, so the position of the ASCII code may change, depending on the size of the data storage area.

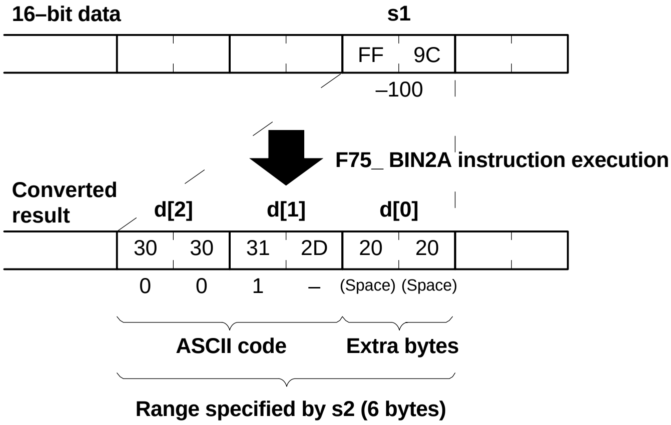
When s2 = 8 (8 bytes)

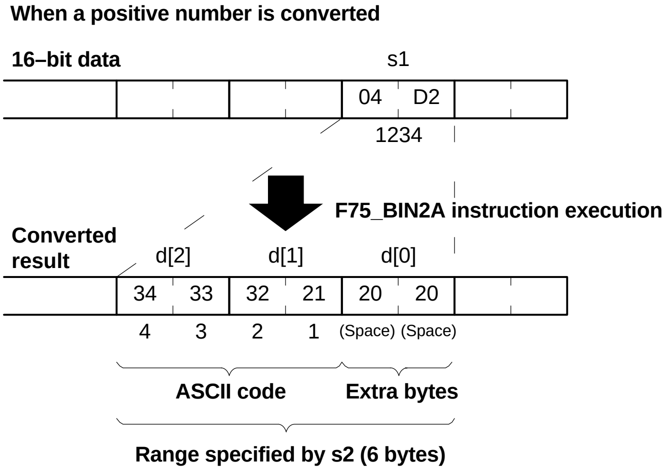


- If the number of bytes of ASCII codes following conversion (including the minus sign) is larger than the number of bytes specified by the **s2**, an operation error occurs. Make sure the sign is taken into consideration when specifying the object of conversion for the **s2**.

The following illustrations show conversions from 16–bit decimal data to ASCII codes.

When a negative number is converted





Decimal characters to express ASCII HEX code:

Decimal characters	ASCII HEX code
SPACE	16#20
–	16#2D
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit area to be converted (source)
s2	INT	specifies number of bytes used to express destination data (ASCII codes)
d	WORD	16–bit area for storing ASCII codes (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

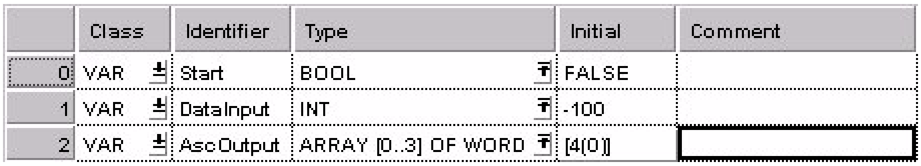
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the number of bytes specified by s2 exceeds the area specified by d. – the data specified by s2 is recognized as “0”.
R9008	%MX0.900.8	for an instant	– the converted result exceeds the area specified by d. – the number of bytes of converted result exceeds the number of bytes specified by s2.

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

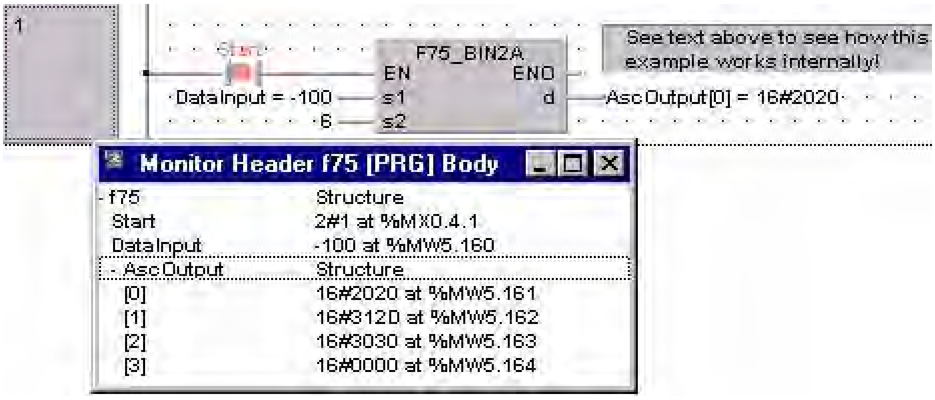


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	DataInput	INT	-100	
2	VAR	AscOutput	ARRAY [0..3] OF WORD	[4(0)]	

Body

When the variable *Start* is set the TRUE, the function is executed. This programming example is based on the example for the conversion of a negative number outlined above. The monitor value icon is activated for both the LD and IL bodies; the monitor header icon is activated for the LD body.

LD



See text above to see how this example works internally!

Monitor Header f75 [PRG] Body

```
- f75             Structure
  Start           2#1 at %MX0.4.1
  DataInput       -100 at %MW5.160
 - AscOutput      Structure
    [0]           16#2020 at %MW5.161
    [1]           16#312D at %MW5.162
    [2]           16#3030 at %MW5.163
    [3]           16#0000 at %MW5.164
```

ST

```
IF start THEN
      F75_BIN2A( s1:= DataInput ,
             s2_Number:= 6 ,
             d_Start=> ASCOutput[0] );
END_IF;
```

F76_A2BIN

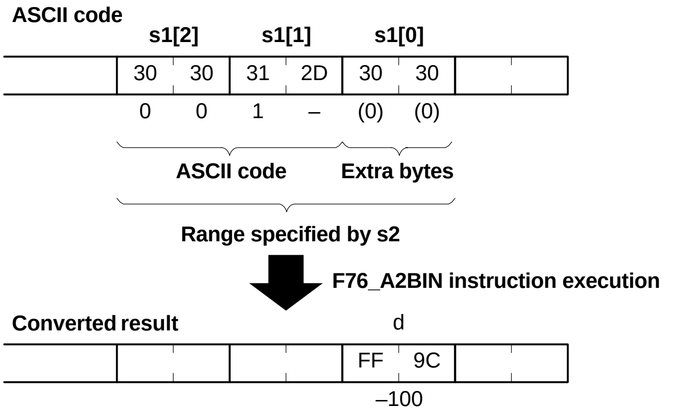
ASCII → 16–bit BIN conversion

Steps	7

Description Converts the ASCII codes that express the decimal digits, starting from the 16-bit area specified by **s1** to 16-bit data as specified by **s2**. The converted result is stored in the area specified by **d**. **s2** specifies the number of source data bytes to be converted using decimal number. (This specification cannot be made with BCD data.)

- The ASCII codes being converted should be stored in the direction of the last address in the specified area.
- If the area specified by **s1** and **s2** is more than that required for the data you want to convert, place “0” (ASCII HEX code: 16#30) or “SPACE” (ASCII HEX code: 16#20) into the extra bytes.
- ASCII codes with signs (such as +: 16#2B and –: 16#2D) are also converted. The + codes can be omitted.

Example of converting an ASCII code indicating a negative number



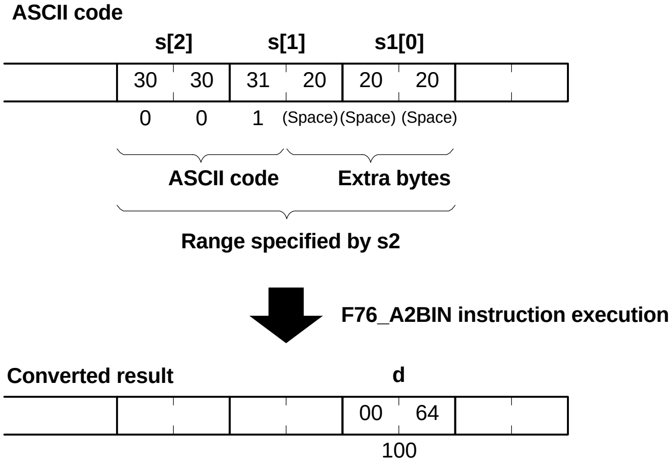
ASCII code

	s1[2]		s1[1]		s1[0]			
	30	30	31	2D	30	30		
	0	0	1	–	(0)	(0)		

Converted result

					d			
					FF	9C		
					–100			

Example of converting an ASCII code indicating a positive number



ASCII code

	s[2]		s[1]		s1[0]			
	30	30	31	20	20	20		
	0	0	1	(Space)	(Space)	(Space)		

Converted result

					d			
					00	64		
					100			

ASCII HEX code to express decimal characters:

ASCII HEX code	Decimal characters
16#20	SPACE
16#2B	+
16#2D	–
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9

Data types

Variable	Data type	Function
s1	WORD	16–bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	INT, WORD	16–bit area for storing converted data (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
s2	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

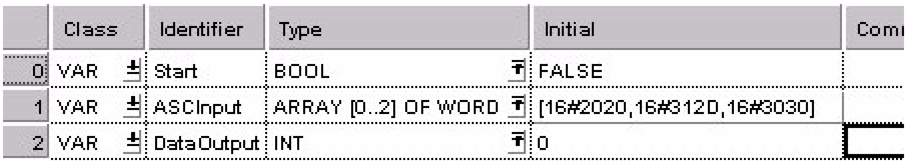
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the number of bytes specified by s2 exceeds the area specified by s1. – the data specified by s2 is recognized as “0”. – the converted result exceeds the 16-bit area specified by d. – ASCII code not corresponding to decimal numbers (0 to 9) or ASCII characters (+, –, and SPACE) is specified.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

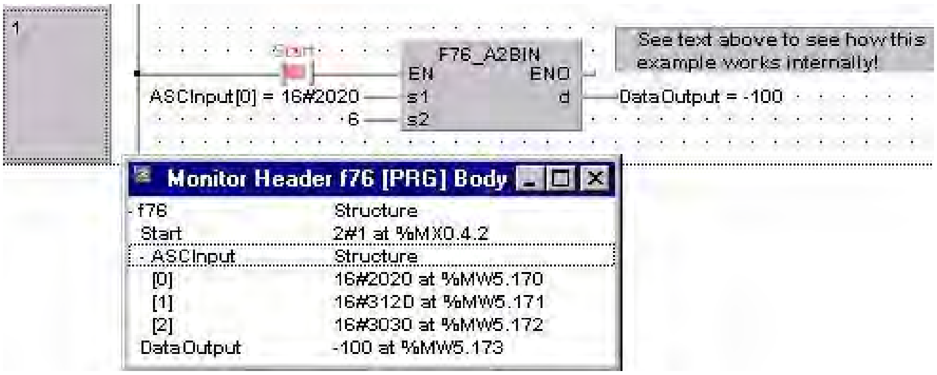


	Class	Identifier	Type	Initial	Comi
0	VAR	Start	BOOL	FALSE	
1	VAR	ASCInput	ARRAY [0..2] OF WORD	[16#2020,16#312D,16#3030]	
2	VAR	DataOutput	INT	0	

Body

When the variable *Start* is set the TRUE, the function is executed. The number of bytes to be converted is entered directly at the contact pin for s2. This programming example is based on the example for the conversion of a negative number outlined above.

LD


1
Start
F76_A2BIN
EN
ENO
ASCInput[0] = 16#2020
s1
d
DataOutput = -100
6
s2
See text above to see how this example works internally!
Monitor Header f76 [PRG] Body

- f76	Structure
Start	2#1 at %MX0.4.2
- ASCInput	Structure
[0]	16#2020 at %MW5.170
[1]	16#312D at %MW5.171
[2]	16#3030 at %MW5.172
DataOutput	-100 at %MW5.173

ST

```
IF start THEN
        F76_A2BIN( s1_Start:= ASCInput[0] ,
                s2_Number:= 6 ,
                d=> DataOutput );
END_IF;
```

F77_DBIN2A

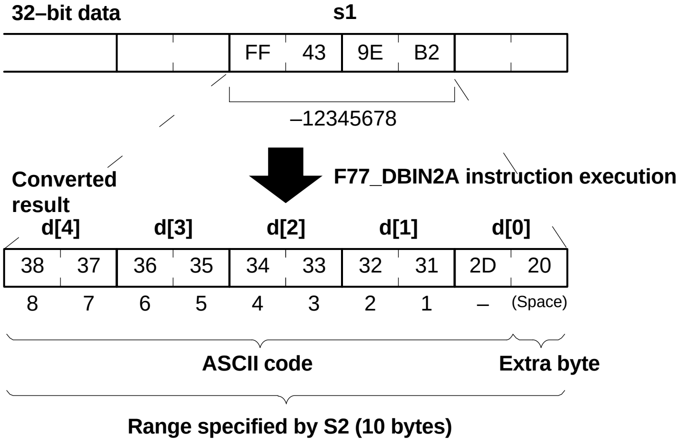
32–bit BIN → ASCII conversion

Steps	11

Description Converts the 32-bit data specified by **s1** to ASCII code that expresses the equivalent decimals. The converted result is stored in the area starting from the 16-bit area specified by **d** as specified by **s2**. **s2** specifies the number of bytes used to express the destination data using decimal.

- When a positive number is converted, the "+" sign is not converted.
- When a negative number is converted, the "–" sign is also converted to ASCII code (ASCII HEX code: 16#2D).
- If the area specified by **s2** is more than that required by the converted data the ASCII code for "SPACE" (ASCII HEX code: 16#20) is stored in the extra area.
- Data is stored in the direction of the last address, so the position of the ASCII code may change depending on the size of the data storage area.
- If the number of bytes of ASCII codes following conversion (including the minus sign) is larger than the number of bytes specified by the **s2**, an operation error occurs. Make sure the sign is taken into consideration when specifying the object of conversion for the **s2**.

Example of converting a negative number from 32–bit decimal format to ASCII codes



Decimal characters to express ASCII HEX code:

Decimal characters	ASCII HEX code
SPACE	16#20
+	16#2B
–	16#2D
0	16#30
1	16#31
2	16#32
3	16#33
4	16#34
5	16#35
6	16#36
7	16#37
8	16#38
9	16#39

Data types

Variable	Data type	Function
s1	DINT, DWORD	32–bit data area to be converted (source)
s2	INT	specifies number of bytes to express destination data (ASCII codes)
d	WORD	16–bit area for storing ASCII codes (destination)

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	x
s2	**WX**	**WY**	**WR**	**WL**	**SV**	**EV**	**DT**	**LD**	**FL**	**dec. or hex.**
	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

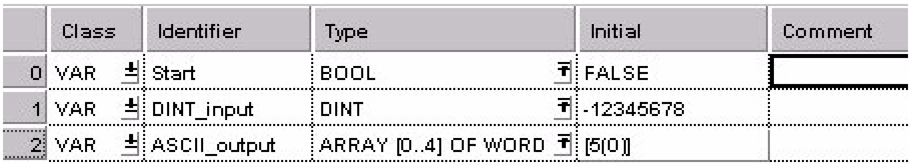
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the number of bytes specified by s2 exceeds the area specified by d. – the data specified by s2 is recognized as “0”. – the converted result exceeds the area specified by d. – the number of bytes of converted result exceeds the number of bytes specified by s2.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

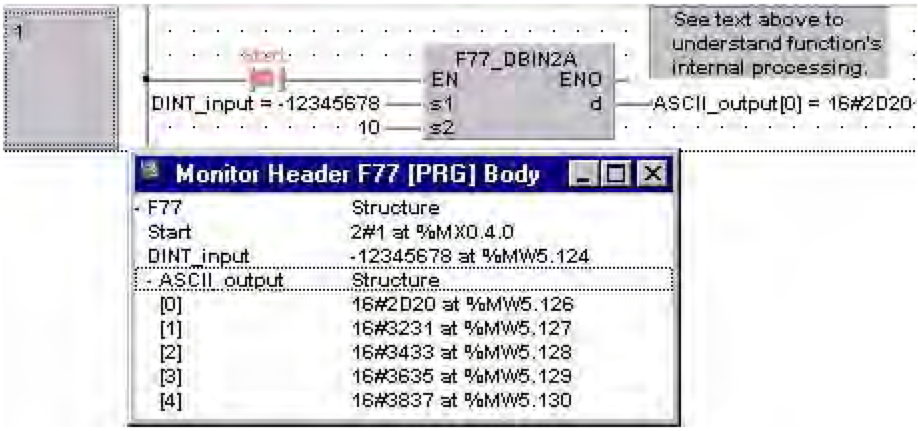


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	DINT_input	DINT	-12345678	
2	VAR	ASCII_output	ARRAY [0..4] OF WORD	[5(0)]	

Body

When the variable *Start* is set to TRUE, the function is executed. The number of bytes to be converted is entered directly at the contact pin for s2. This programming example is based on the example for the conversion of a negative number outlined above.

LD


1
Start
F77_DBIN2A
EN
ENO
DINT_input = -12345678
s1
d
10
s2
ASCII_output[0] = 16#2D20
See text above to understand function's internal processing.

Monitor Header F77 [PRG] Body

```
- F77               Structure
  Start             2#1 at %MX0.4.0
  DINT_input        -12345678 at %MW5.124
- ASCII_output      Structure
    [0]             16#2D20 at %MW5.126
    [1]             16#3231 at %MW5.127
    [2]             16#3433 at %MW5.128
    [3]             16#3635 at %MW5.129
    [4]             16#3837 at %MW5.130
```

ST

```
IF start THEN
      F77_DBIN2A( s1:= DINT_input ,
             s2_Number:= 10 ,
             d_Start=> ASCII_output[0] );
END_IF;
```

F78_DA2BIN

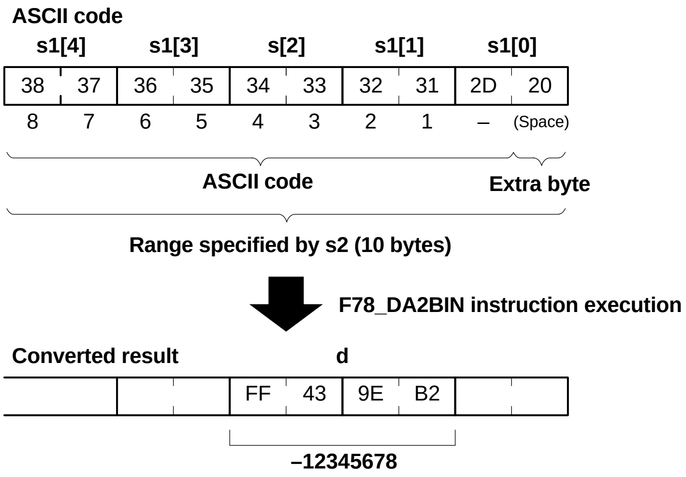
ASCII → 32–bit BIN conversion

Steps	11

Description Converts ASCII code that expresses the decimal digits, starting from the 16-bit area specified by **s1** to 32-bit data as specified by **s2**. The converted result is stored in the area starting from the 16-bit area specified by **d**. **s2** specifies the number of bytes used to express the destination data using decimals.

- The ASCII codes being converted should be stored in the direction of the last address in the specified area.
- If the area specified by **s1** and **s2** is more than that required by the data you want to convert, place “0” (ASCII HEX code: 16#30) or “SPACE” (ASCII HEX code: 16#20) in the extra bytes.
- ASCII codes with signs (such as +: 16#2B and –: 16#2D) are also converted. The + codes can be omitted.

Example of converting an ASCII code indicating a negative number


ASCII code
s1[4]
s1[3]
s[2]
s1[1]
s1[0]
38 37 36 35 34 33 32 31 2D 20
8 7 6 5 4 3 2 1 – (Space)
ASCII code
Extra byte
Range specified by s2 (10 bytes)
F78_DA2BIN instruction execution
Converted result
d
FF 43 9E B2
–12345678

ASCII HEX code to express decimal characters:

ASCII HEX code	Decimal characters
16#20	SPACE
16#2B	+
16#2D	–
16#30	0
16#31	1
16#32	2
16#33	3
16#34	4
16#35	5
16#36	6
16#37	7
16#38	8
16#39	9

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F78	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	WORD	starting 16–bit area for ASCII code (source)
s2	INT	specifies number of source data bytes to be converted
d	DINT, DWORD	area for 32–bit data storage (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	–
s2	x	x	x	x	x	x	x	x	x	x
d	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

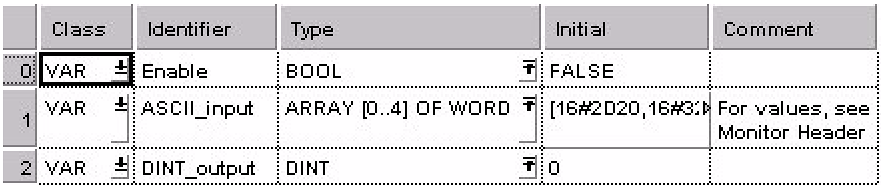
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the number of bytes specified by s2 exceeds the area specified by s1. – the data specified by s2 is recognized as “0”. – the converted result exceeds the area specified by d.
R9008	%MX0.900.8	for an instant	– the converted result exceeds the 32-bit area. – ASCII code not corresponding to decimal numbers (0 to 9) or ASCII characters (+, –, and SPACE) is specified.

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

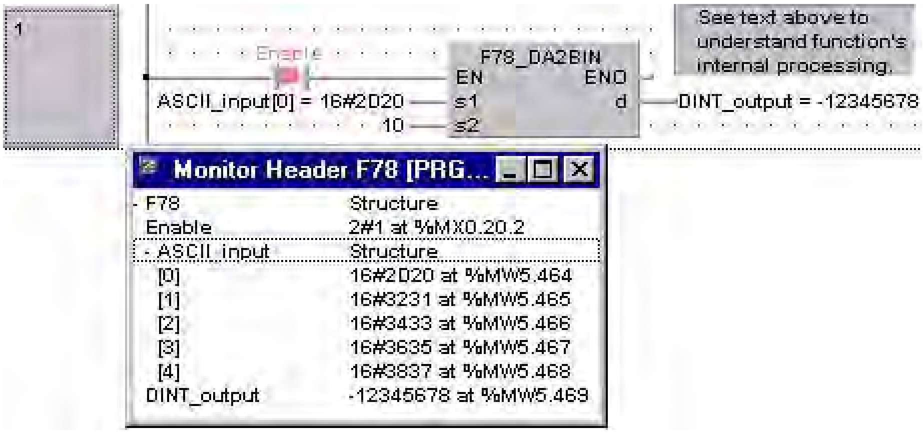


	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	ASCII_input	ARRAY [0..4] OF WORD	[16#2D20,16#3	For values, see Monitor Header
2	VAR	DINT_output	DINT	0	

Body

When the variable *Enable* is set to TRUE, the function is executed. The number of bytes to be converted is entered directly at the contact pin for s2. This programming example is based on the example for the conversion of a negative number outlined above.

LD


1
Enable
F78_DA2BIN
EN
ENO
s1
d
s2
ASCII_input[0] = 16#2D20
10
DINT_output = -12345678
See text above to understand function's internal processing.
Monitor Header F78 [PRG...

```
- F78            Structure
  Enable         2#1 at %MX0.20.2
- ASCII_input    Structure
    [0]          16#2D20 at %MW5.464
    [1]          16#3231 at %MW5.465
    [2]          16#3433 at %MW5.466
    [3]          16#3635 at %MW5.467
    [4]          16#3837 at %MW5.468
  DINT_output    -12345678 at %MW5.469
```

ST

```
IF start THEN
      F78_DA2BIN( s1_Start:= ASCII_input[0] ,
             s2_Number:= 10 ,
             d=> DINT_output );
END_IF;
```

F80_BCD

16–bit decimal → 4–digit BCD conversion

Steps	5

Description

Converts the 16–bit binary data specified by **s** to the BCD code that expresses 4–digit decimals if the trigger **EN** is in the ON–state. The converted data is stored in **d**. The binary data that can be converted to BCD code are in the range of 0 (0 hex) to 9999 (270F hex).

Source [s]: 16

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
Decimal	16			

↓ **Conversion (to BCD code)**

Destination [d]: 16#16 (BCD)

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
BCD code	0 0 0 0	0 0 0 0	0 0 0 1	0 1 1 0
BCD Hex code	0	0	1	6

Data types

Variable	Data type	Function
s	INT, WORD	binary data (source), range: 0 to 9999
d	WORD	16–bit area for 4–digit BCD code (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

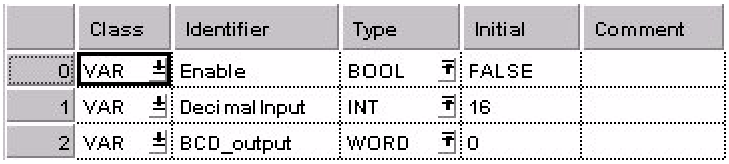
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	16-bit binary data outside the range of 0 (16#0) to 9999 (16#270F) is converted.
R9008	%MX0.900.8	for an instant	

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

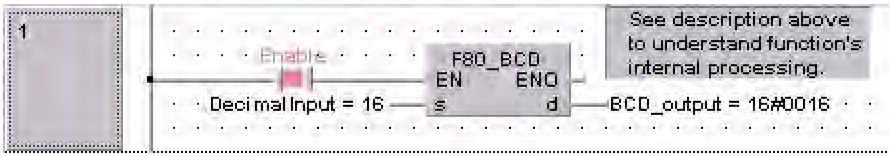


	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	DecimalInput	INT	16	
2	VAR	BCD_output	WORD	0	

Body

When the variable *Enable* is set to TRUE, the function is executed. The decimal value in *DecimalInput* is converted to a BCD hexadecimal value and stored in the variable *BCD_output*.

LD



See description above to understand function's internal processing.

ST

```
IF Enable THEN
      F80_BCD(DecimalInput, BCD_output);
END_IF;
```

F81_BIN

4–digit BCD → 16–bit decimal conversion

Steps	5

Description Converts the BCD code that expresses 4–digit decimals specified by **s** to 16–bit binary data if the trigger **EN** is in the ON–state. The converted result is stored in the area specified by **d**.

Source [s]: 16#15 (BCD)

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
BCD code	0 0 0 0	0 0 0 0	0 0 0 1	0 1 0 1
BCD Hex code	0	0	1	5

↓ **Conversion (to binary data)**

Destination [d]: 15

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1
Decimal	15			

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F81	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	WORD	16–bit area for 4–digit BCD data (source)
d	INT, WORD	16–bit area for storing 16–bit binary data (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

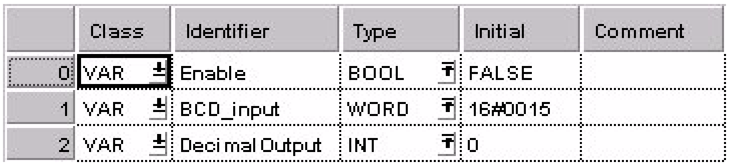
x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the data specified by s is not BCD data.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

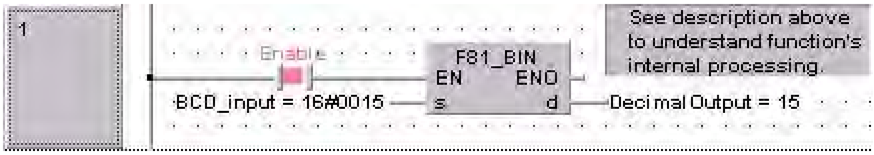
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	BCD_input	WORD	16#0015	
2	VAR	Deci mal Output	INT	0	

Body When the variable *Enable* is set to TRUE, the function is executed. The BCD value assigned to the variable *BCD_input* is converted to a decimal value and stored in the variable *DecimalOutput*. The monitor value icon is activated for both the LD and IL bodies.

LD



See description above to understand function's internal processing.

ST

```
IF Enable THEN
      F81_BIN(BCD_Input, DecimalOutput);
END_IF;
```

F82_DBCD 32–bit decimal → 8–digit BCD conversion

Steps	7

Description Converts the 32–bit binary data specified by **s** to the BCD code that expresses 8–digit decimals if the trigger **EN** is in the ON–state. The converted data is stored in **d**. The binary data that can be converted to BCD code are in the range of 0 (0 hex) to 99,999,999 (5F5E0FF hex).

Source (s): 72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	0 0 0 0	0 1 0 0	0 1 0 1	0 1 1 1	0 0 0 0	0 1 0 0	1 1 0 1	0 0 1 0
Decimal	72811730							

Higher 16-bit area **Lower 16-bit area**

Destination (d): 16#72811730 (BCD)

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
BCD code	0 1 1 1	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 1	0 1 1 1	0 0 1 1	0 0 0 0
BCD Hex code	7	2	8	1	1	7	3	0

Higher 16-bit area **Lower 16-bit area**

Data types

Variable	Data type	Function
s	DINT, DWORD	binary data (source), range: 0 to 99,999,999
d	DWORD	32–bit area for 8–digit BCD code (destination)

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

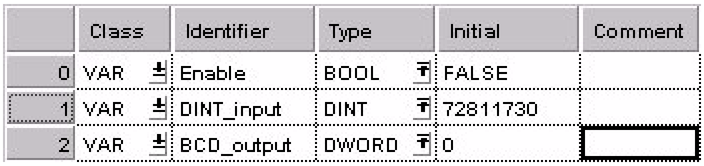
x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	32-bit data specified by s outside the range of 0 (16#0) to 99999999 (16#5F5E0FF) is converted.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

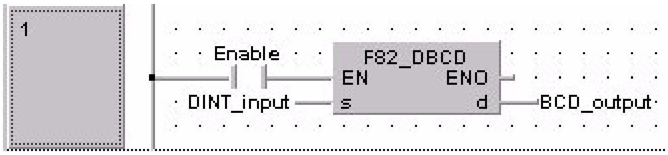
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	DINT_input	DINT	72811730	
2	VAR	BCD_output	DWORD	0	

Body When the variable *Enable* is set to TRUE, the function is executed. The decimal value in *DINT_input* is converted to a BCD hexadecimal value and stored in the variable *BCD_output*. You may also assign a decimal, binary (prefix 2#), or hexadecimal (prefix 16#) value directly at the contact pin for s.

LD



ST

```
IF Enable THEN
      F82_DBCD(DINT_input, BCD_output);
END_IF;
```

F83_DBIN 8–digit BCD → 32–bit decimal conversion

Steps	7

Description Converts the BCD code that expresses 8–digit decimals specified by **s** to 32–bit binary data if the trigger **EN** is in the ON–state. The converted result is stored in the area specified by **d**.

Source (s): 16#72811730 (BCD)

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
BCD code	0 1 1 1	0 0 1 0	1 0 0 0	0 0 0 1	0 0 0 1	0 1 1 1	0 0 1 1	0 0 0 0
BCD Hex code	7	2	8	1	1	7	3	0

Higher 16-bit area **Lower 16-bit area**

Destination (d): 72811730

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Binary data	0 0 0 0	0 1 0 0	0 1 0 1	0 1 1 1	0 0 0 0	0 1 0 0	1 1 0 1	0 0 1 0
Decimal	72811730							

Higher 16-bit area **Lower 16-bit area**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F83	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DWORD	area for 8–digit BCD data (source)
d	DINT, DWORD	32–bit area for storing 32–bit data (destination)

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

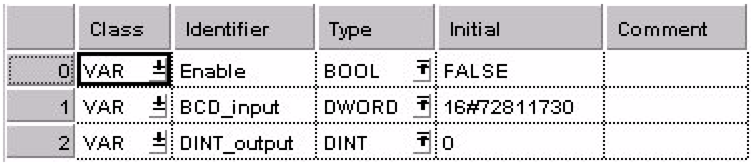
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the data specified by s is not BCD data.
R9008	%MX0.900.8	for an instant	

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

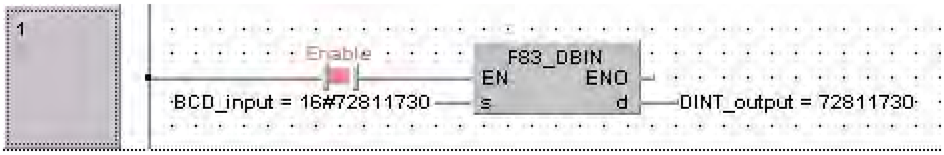


	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	BCD_input	DWORD	16#72811730	
2	VAR	DINT_output	DINT	0	

Body

When the variable *Enable* is set to TRUE, the function is executed. The BCD value assigned to the variable *BCD_input* is converted to a decimal value and stored in the variable *DINT_output*.

LD



ST

```
IF Enable THEN
      F83_DBIN(BCD_input, DINT_Output);
END_IF;
```

F84_INV

16–bit data invert (one's complement)

Steps	3

Description

Inverts each bit (0 or 1) of the 16–bit data specified by **d** if the trigger **EN** is in the ON–state. The inverted result is stored in the 16–bit area specified by **d**. This instruction is useful for controlling an external device that uses negative logic operation.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F84	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area to be inverted

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

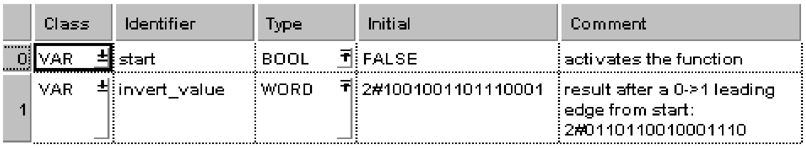
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

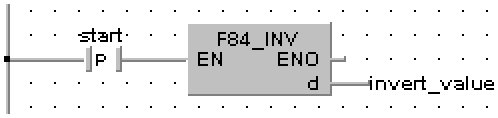


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	invert_value	WORD	2#1001001101110001	result after a 0->1 leading edge from start: 2#0110110010001110

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F84_INV(invert_value);
END_IF;
```

F85_NEG

16–bit data two’s complement

Steps	3

Description Gets the two’s complement of 16–bit data specified by **d** if the trigger **EN** is in the ON–state. The two’s complement of the original 16–bit data is stored in **d**.

Two’s complement: A number system used to express positive and negative numbers in binary. In this system, the number becomes negative if the most significant bit (MSB) of data is 1. The two’s complement is obtained by inverting all bits and adding 1 to the inverted result.

This instruction is useful for inverting the sign of 16–bit data from positive to negative or from negative to positive.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F85	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area for storing original data and its two’s complement

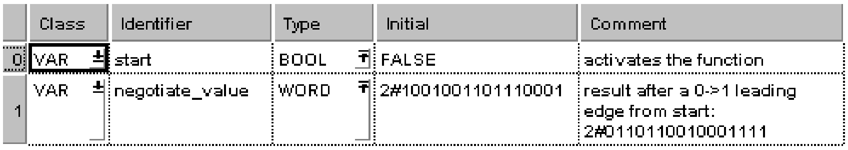
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

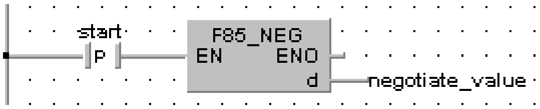
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negotiate_value	WORD	2#1001001101110001	result after a 0->1 leading edge from start: 2#0110110010001111

Body When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F85_NEG(negotiate_value);
END_IF;
```

F86_DNEG

32–bit data two’s complement

Steps	3

Description

Gets the two’s complement of 32–bit data specified by **d** if the trigger **EN** is in the ON–state. The two’s complement of the original 32–bit data is stored in **d**.

Two’s complement: A number system used to express positive and negative numbers in binary. In this system, the number becomes negative if the most significant bit (MSB) of data is 1. The two’s complement is obtained by inverting all bits and adding 1 to the inverted result.

This instruction is useful for inverting the sign of 16–bit data from positive to negative or from negative to positive.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F86	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	DINT, DWORD	32–bit area for storing original data and its two’s complement

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–

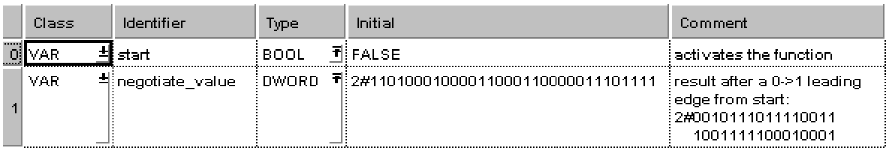
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

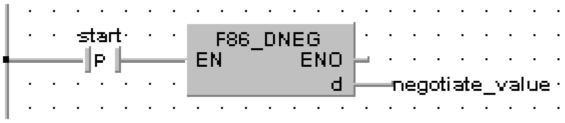


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	negotiate_value	DWORD	2#11010001000011000110000011101111	result after a 0->1 leading edge from start: 2#0010111011110011 1001111100010001

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F86_DNEG(negotiate_value);
END_IF;
```

F89_EXT

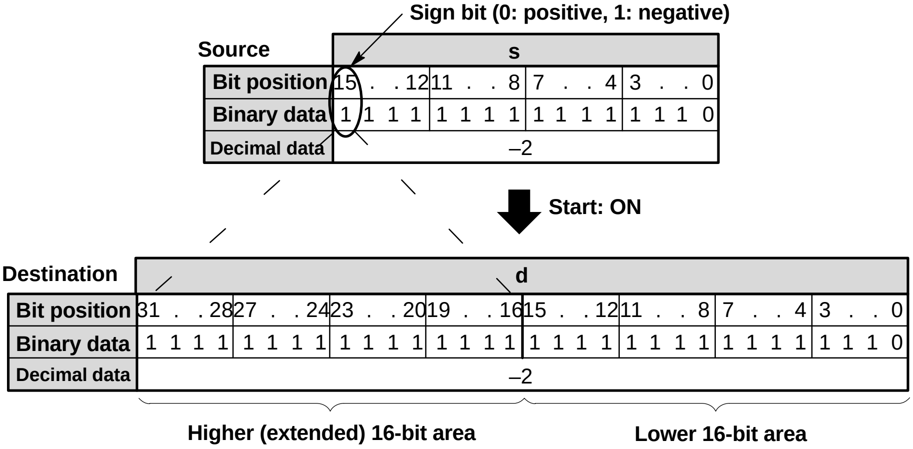
16–bit data sign extension

Steps 3

Description

16–bit data is converted to 32–bit data without signs and values being changed. F89 copies the sign bit of the 16–bit data specified in **s** to all the bits of the higher 16–bit area (extended 16–bit area) in **d**.

If the sign bit (bit position 15) of the 16–bit data specified by **s** is 0, all higher 16 bits in the variable assigned to **d** will be 0. If the sign bit of **s** is 1, the higher 16 bits of **d** will be 1.


Sign bit (0: positive, 1: negative)
Start: ON
Higher (extended) 16-bit area
Lower 16-bit area

Source	s			
Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
Decimal data	–2			

Destination	d							
Bit position	31 . . 28	27 . . 24	23 . . 20	19 . . 16	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
Decimal data	–2							

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F89	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	16–bit source data area, bit 15 is sign bit
d	DINT, DWORD	32–bit destination area, s copied to lower 16 bits, higher 16 bits filled with sign bit of s

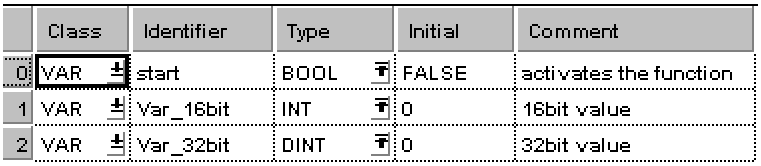
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	–	x	x	x	x	x	x	x	x	–
d	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

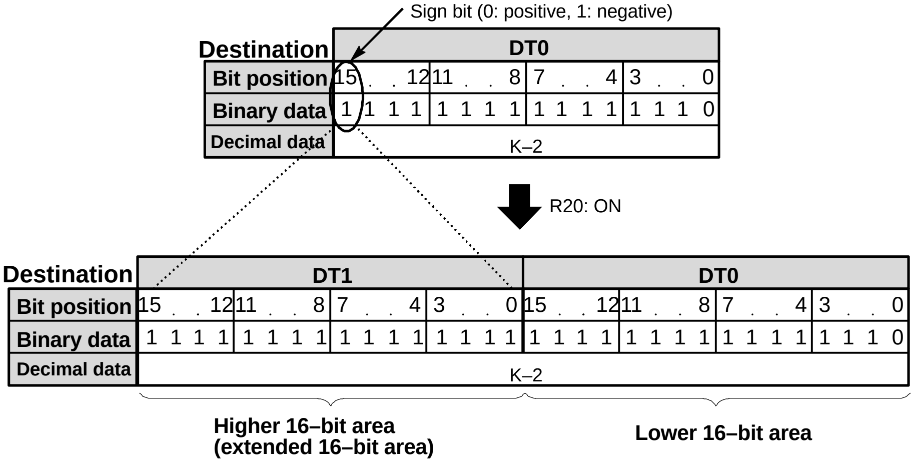
Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	Var_16bit	INT	0	16bit value
2	VAR	Var_32bit	DINT	0	32bit value

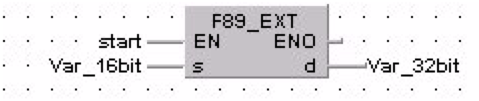
Body When the variable *start* is set to TRUE, the function is executed.



Destination	DT0			
Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
Decimal data	K–2			

Destination	DT1				DT0			
Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0	15 . . 12	11 . . 8	7 . . 4	3 . . 0
Binary data	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 0
Decimal data	K–2							

LD



ST

```
IF start THEN
      F89_EXT(Var_16bit, Var_32bit);
END_IF;
```

F90_DECO

Decode hexadecimal –> bit state

Steps	7

Description

Decodes the contents of 16–bit data specified by **s** according to the contents of **n** if the trigger **EN** is in the ON–state. The decoded result is stored in the area starting with the 16–bit area specified by **d**.

n specifies the starting bit position and the number of bits to be decoded using hexadecimal data:
Bit no. 0 to 3: number of bits to be decoded
Bit no. 8 to 11: starting bit position to be decoded
(The bits nos. 4 to 7 and 12 to 15 are invalid.)
e.g. when **n** = 16#0404, four bits beginning at bit position four are decoded.

Relationship between number of bits and occupied data area for decoded result:

Number of bits to be decoded	Data area required for the result	Valid bits in the area for the result
1	1-word	2-bit*
2	1-word	4-bit*
3	1-word	8-bit*
4	1-word	16-bit
5	2-word	32-bit
6	4-word	64-bit
7	8-word	128-bit
8	16-word	256-bit

*Invalid bits in the data area required for the result are set to 0.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F90	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	source 16–bit area or equivalent constant to be decoded
n	INT, WORD	control data to specify the starting bit position and number of bits to be decoded
d	INT, WORD	starting 16–bit area for storing decoded data (destination)

The variables **s**, **n** and **d** have to be of the same data type.

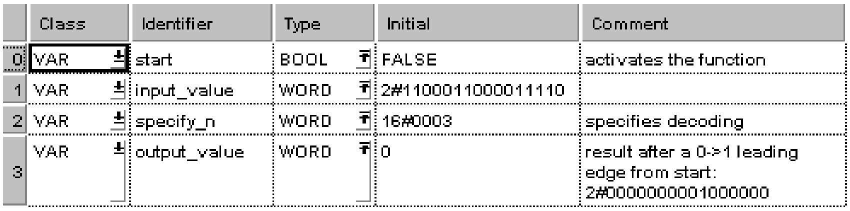
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s, n	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

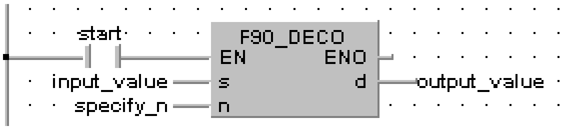
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#1100011000011110	
2	VAR	specify_n	WORD	16#0003	specifies decoding
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000001000000

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F90_DECO( s:= input_value ,
             n:= specify_n ,
             d=> output_value );
END_IF;
```

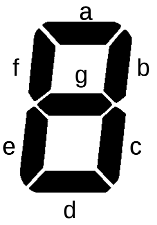
F91_SEGT

16–bit data 7–segment decode

Steps	3

Description Converts the 16–bit equivalent constant or 16–bit data specified by **s** to 4–digit data for 7–segment indication if the trigger **EN** is in the ON–state. The converted data is stored in the area starting with the 16–bit area specified by **d**. The data for 7–segment indication occupies 8 bits (1 byte) to express 1 digit.

7–segment conversion table:

One digit data to be converted		8-bit data for 7-segment indication								7-segment indication	Organization of 7-segment indication
Hexadecimal	Binary	╱	g	f	e	d	c	b	a		
16#0	0 0 0 0	0	0	1	1	1	1	1	1	0	
16#1	0 0 0 1	0	0	0	0	0	1	1	0	1	
16#2	0 0 1 0	0	1	0	1	0	0	1	1	2	
16#3	0 0 1 1	0	1	0	0	1	1	1	1	3	
16#4	0 1 0 0	0	1	1	0	0	1	1	0	4	
16#5	0 1 0 1	0	1	1	0	1	1	0	1	5	
16#6	0 1 1 0	0	1	1	1	1	1	0	1	6	
16#7	0 1 1 1	0	0	1	0	0	1	1	1	7	
16#8	1 0 0 0	0	1	1	1	1	1	1	1	8	
16#9	1 0 0 1	0	1	1	0	1	1	1	1	9	
16#A	1 0 1 0	0	1	1	1	0	1	1	1	A	
16#B	1 0 1 1	0	1	1	1	1	1	0	0	b	
16#C	1 1 0 0	0	0	1	1	1	0	0	0	C	
16#D	1 1 0 1	0	1	0	1	1	1	1	0	d	
16#E	1 1 1 0	0	1	1	1	1	0	0	1	E	
16#F	1 1 1 1	0	1	1	1	0	0	0	1	F	

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F91	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	16–bit area or equivalent constant to be converted to 7–segment indication (source)
d	DINT, DWORD	32–bit area for storing 4–digit data for 7–segment indication (destination)

Operands

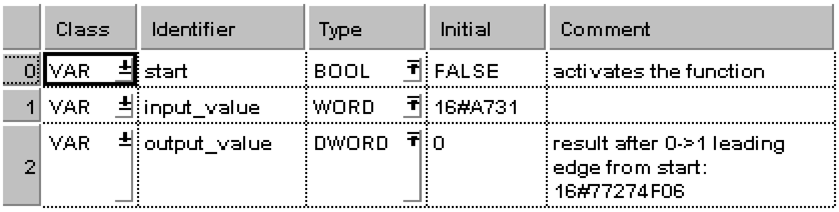
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

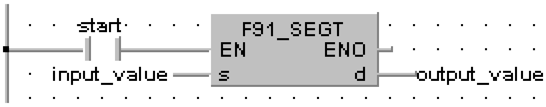


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	16#A731	
2	VAR	output_value	DWORD	0	result after 0->1 leading edge from start: 16#77274F06

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F91_SEGT(input_value, output_value);
END_IF;
```

F92_ENCO

Encode bit state –> hexadecimal

Steps 7

Description

Encodes the contents of data specified by **s** according to the contents of **n** if the trigger **EN** is in the ON–state. The encoded result is stored in the 16–bit area specified by **d** starting with the specified bit position. Invalid bits in the area specified for the encoded result are set to 0.

n specifies the starting bit position of destination data **d** and the number of bits to be encoded using hexadecimal data:
Bit no. 0 to 3: number of bits to be encoded
Bit no. 8 to 11: starting bit position of destination data to be encoded
(The bit nos. 4 to 7 and 12 to 15 are invalid.)
e.g. **n** = 16#0005

Number of bits to be encoded: 2^5 = 32 bits
Starting bit position to be encoded for destination data: bit position 0

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F92	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	starting 16–bit area to be encoded (source)
n	INT, WORD	control data to specify the starting bit position and number of bits to be encoded
d	INT, WORD	16–bit area for storing encoded data (destination)

The variables **s**, **n** and **d** have to be of the same data type.

- **Put at least one bit into the area to be checked to avoid an error message from the PLC.**
- **When several bits are set, the uppermost bit is evaluated.**

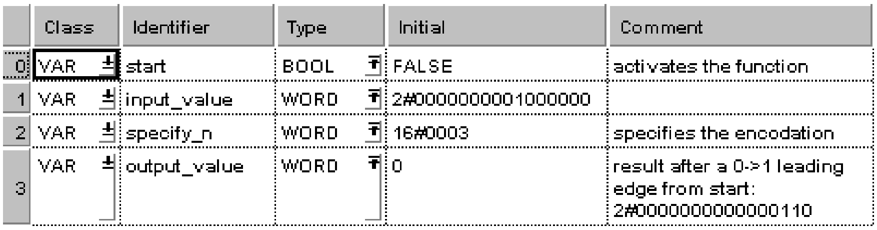
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

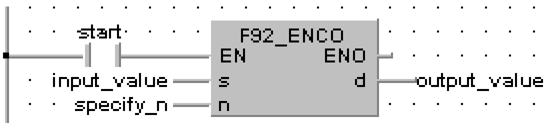
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	WORD	2#0000000001000000	
2	VAR	specify_n	WORD	16#0003	specifies the encodation
3	VAR	output_value	WORD	0	result after a 0->1 leading edge from start: 2#0000000000000110

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F92_ENCO( s:= input_value ,
             n:= specify_n ,
             d=> output_value );
END_IF;
```

F93_UNIT

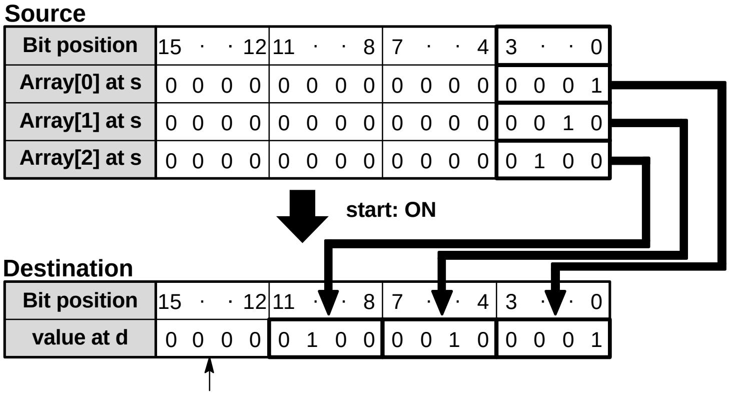
16–bit data combine

Steps	7

Description

Extracts each lower 4 bits (bit position 0 to 3) starting with the 16–bit area specified by **s** and combines the extracted data into 1 word if the trigger **EN** is in the ON–state. The result is stored in the 16–bit area specified by **d**. **n** specifies the number of data to be extracted. The range of **n** is 0 to 4.

The programming example provided below can be envisioned thus:



Source

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Array[0] at s	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1
Array[1] at s	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 0
Array[2] at s	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
value at d	0 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1

Bit positions 12 to 15 are filled with 0s.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F93	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	WORD	starting 16–bit area to be extracted (source)
n	INT	specifies number of data to be extracted
d	WORD	16–bit area for storing combined data (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

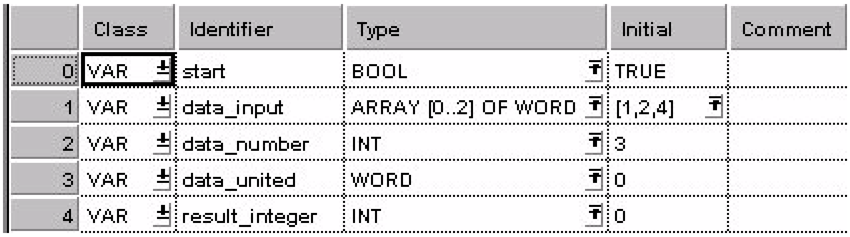
x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the area specified using the index modifier exceeds the limit
R9008	%MX0.900.8	for an instant	– the value at n ≥ 5

Example In this example the function F93_UNIT is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

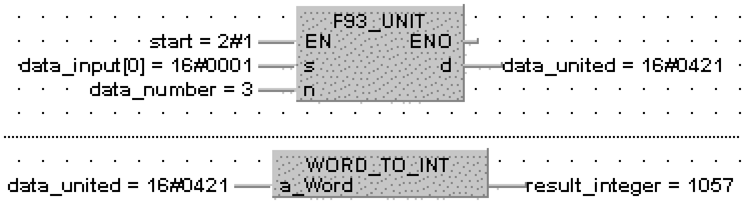
POU header In the POU header, all input and output variables are declared that are used for programming this function.



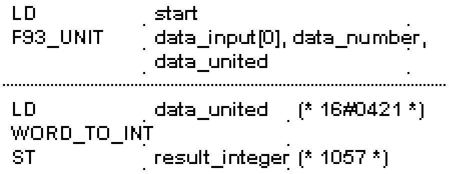
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	TRUE	
1	VAR	data_input	ARRAY [0..2] OF WORD	[1,2,4]	
2	VAR	data_number	INT	3	
3	VAR	data_united	WORD	0	
4	VAR	result_integer	INT	0	

Body When the variable *start* is set to TRUE, the function is carried out. The binary values in the illustration on the previous page serve as the array values in *data_input*. In this example, variables are declared in the POU header. However, you may assign constants directly at the input function’s contact pins instead.

LD In this example, the view icon was activated so you can see the results immediately.



IL



```
LD            start
F93_UNIT      data_input[0], data_number,
              data_united
------------------------------------------
LD            data_united   (* 16#0421 *)
WORD_TO_INT
ST            result_integer (* 1057 *)
```

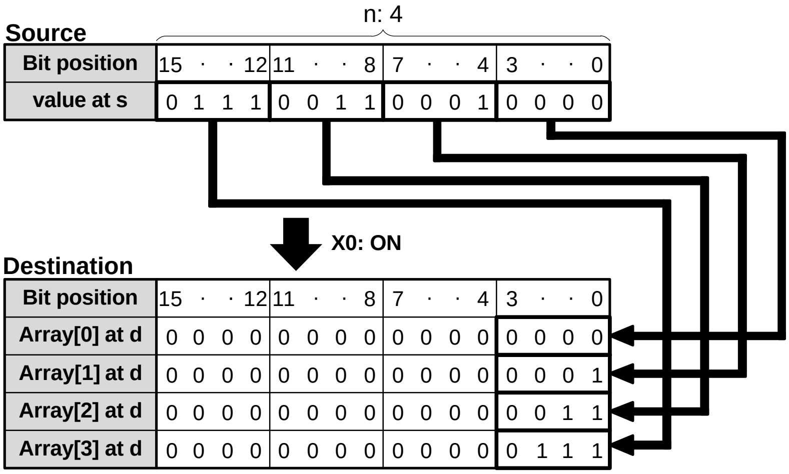
F94_DIST

16–bit data distribution

Steps	7

Description Divides the 16–bit data specified by **s** into 4–bit units and distributes the divided data into the lower 4 bits (bit position 0 to 3) of 16–bit areas starting with **d** if the trigger **EN** is in the ON–state. **n** specifies the number of data to be divided. The range of **n** is 0 to 4). When 0 is specified by **n**, this instruction is not executed.

The programming example provided below can be envisioned thus:



Source

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
value at s	0 1 1 1	0 0 1 1	0 0 0 1	0 0 0 0

Destination

Bit position	15 · · 12	11 · · 8	7 · · 4	3 · · 0
Array[0] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
Array[1] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1
Array[2] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
Array[3] at d	0 0 0 0	0 0 0 0	0 0 0 0	0 1 1 1

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F94	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	WORD	16–bit area or equivalent constant to be divided (source)
n	INT	specifies number of data to be divided
d	WORD	starting 16–bit area for storing divided data (destination)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s, n	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

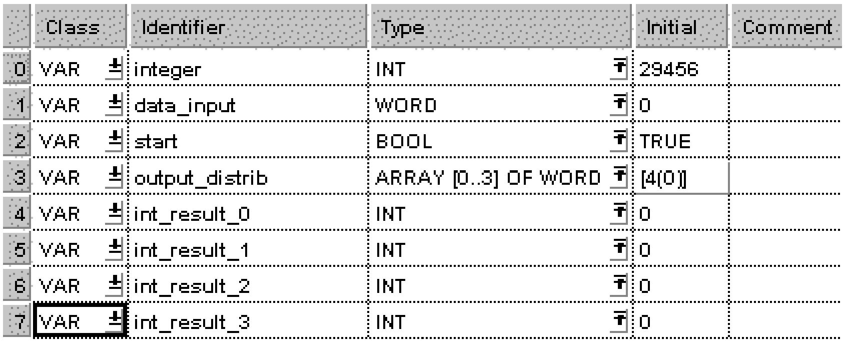
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the area specified using the index modifier exceeds the limit – the value at n ≥ 5 – the last area for the result exceeds the limit
R9008	%MX0.900.8	for an instant	

Example In this example the function F94_DIST is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



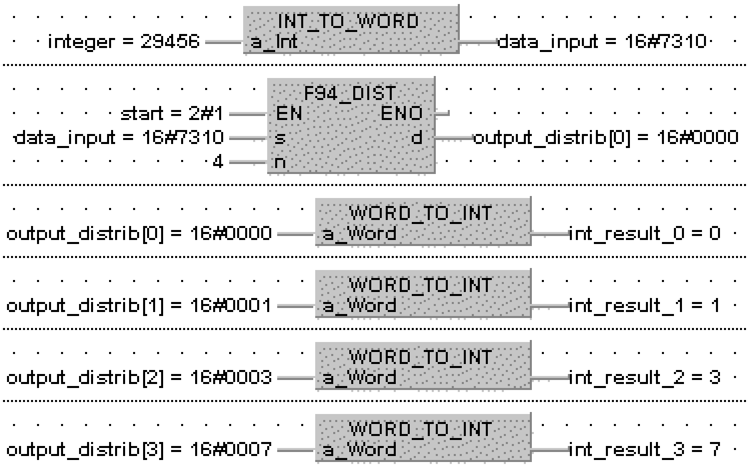
	Class	Identifier	Type	Initial	Comment
0	VAR	integer	INT	29456	
1	VAR	data_input	WORD	0	
2	VAR	start	BOOL	TRUE	
3	VAR	output_distrib	ARRAY [0..3] OF WORD	[4(0)]	
4	VAR	int_result_0	INT	0	
5	VAR	int_result_1	INT	0	
6	VAR	int_result_2	INT	0	
7	VAR	int_result_3	INT	0	

Body

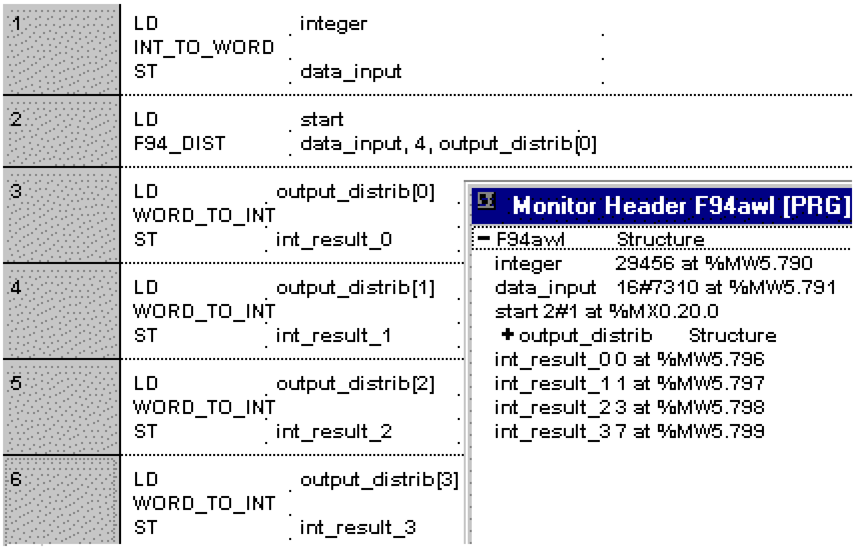
When the variable *start* is set to TRUE, the function is carried out. The binary values in the illustration on the previous page serve as the values calculated. In this example, variables are declared in the POU header. Also, a constant value of 4 is assigned directly at the contact pin for **n**.

LD

In this example, the view icon was activated so you can see the results immediately.



IL Activating the Monitor Header window (Monitor –> Monitor Header) while online also allows you to see results immediately.



```
1   LD              integer
    INT_TO_WORD
    ST              data_input

2   LD              start
    F94_DIST        data_input, 4, output_distrib[0]

3   LD              output_distrib[0]
    WORD_TO_INT
    ST              int_result_0

4   LD              output_distrib[1]
    WORD_TO_INT
    ST              int_result_1

5   LD              output_distrib[2]
    WORD_TO_INT
    ST              int_result_2

6   LD              output_distrib[3]
    WORD_TO_INT
    ST              int_result_3
```

```
- F94awl        Structure
  integer       29456 at %MW5.790
  data_input    16#7310 at %MW5.791
  start 2#1 at %MX0.20.0
  + output_distrib      Structure
  int_result_0 0 at %MW5.796
  int_result_1 1 at %MW5.797
  int_result_2 3 at %MW5.798
  int_result_3 7 at %MW5.799
```

F95_ASC

Character → ASCII transfer

Steps	15

Description Converts the character constants specified by **s** to ASCII code. The converted ASCII code is stored in 6 words starting from the 16-bit area specified by **d**.

[s]	Character constants	ABC1230 DEF

[d]	Data register	d[5]		d[4]		d[3]		d[2]		d[1]		d[0]	
	ASCII HEX code	2 0	4 6	4 5	4 4	2 0	3 0	3 3	3 2	3 1	4 3	4 2	4 1
	ASCII character		F	E	D		0	3	2	1	C	B	A

SPACE

If the number of character constants specified by s **is less than 12, the ASCII code 16#20 (SPACE) is stored in the extra destination area, e.g. s = '12345', d[0] = 3231, d[1] = 3433, d[2] = 2034, d[3] – d[5] = 2020.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F95	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	constant, no variable possible	Character constants, max. 12 letters (source).
d	WORD	Starting 16–bit area for storing 6–word ASCII code (destination).

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	character
s	–	–	–	–	–	–	–	–	–	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the last area for ASCII code exceeds the limit (6 words: six 16–bit areas).
R9008	%MX0.900.8	for an instant	

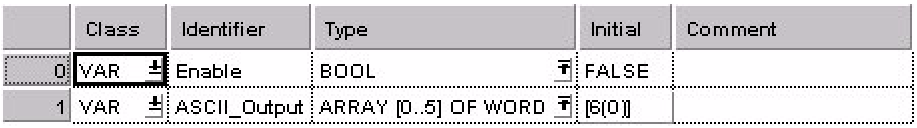
ASCII
HEX code

									b7								
									b6	0	0	0	0	1	1	1	1
									b5	0	0	1	1	0	0	1	1
									b4	0	1	0	1	0	1	0	1
										Most significant digit							
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0		ASCII HEX code	0	1	2	3	4	5	6	7
				0	0	0	0	Least significant digit	0	NUL	DLE	SPACE	0	@	P		p
				0	0	0	1		1	SOH	DC1	!	1	A	Q	a	q
				0	0	1	0		2	STX	DC2	”	2	B	R	b	r
				0	0	1	1		3	ETX	DC3	#	3	C	S	c	s
				0	1	0	0		4	EOT	DC4	$	4	D	T	d	t
				0	1	0	1		5	ENQ	NAK	%	5	E	U	e	u
				0	1	1	0		6	ACK	SYN	&	6	F	V	f	v
				0	1	1	1		7	BEL	ETB	’	7	G	W	g	w
				1	0	0	0		8	BS	CAN	(	8	H	X	h	x
				1	0	0	1		9	HT	EM	)	9	I	Y	i	y
				1	0	1	0		A	LF	SUB	*	:	J	Z	j	z
				1	0	1	1		B	VT	ESC	+	;	K	[	k	{
				1	1	0	0		C	FF	FS	,	<	L	\	l	|
				1	1	0	1		D	CR	GS	–	=	M	]	m	}
				1	1	1	0		E	SO	RS	.	>	N	^	n	~
				1	1	1	1		F	SI	US	/	?	O	_	o	DEL

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

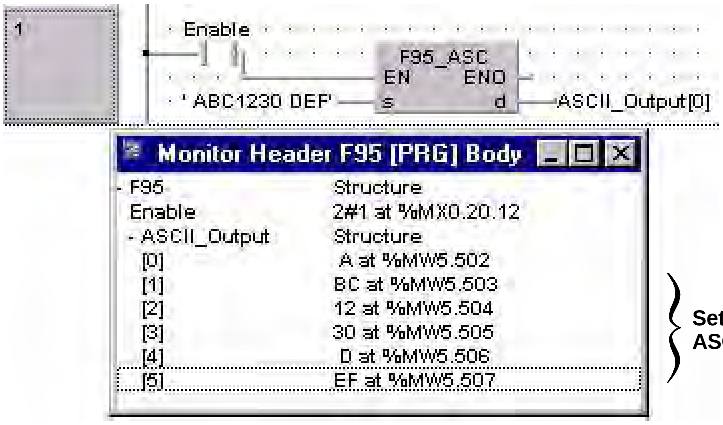


	Class	Identifier	Type	Initial	Comment
0	VAR	Enable	BOOL	FALSE	
1	VAR	ASCII_Output	ARRAY [0..5] OF WORD	[6(0)]	

Body

When the variable *Enable* is enabled, the character constants entered at the input s are converted to ASCII code and stored in the variable *ASCII_Output*.

LD



```
- F95               Structure
  Enable            2#1 at %MX0.20.12
  - ASCII_Output    Structure
    [0]              A at %MW5.502
    [1]             BC at %MW5.503
    [2]             12 at %MW5.504
    [3]             30 at %MW5.505
    [4]              D at %MW5.506
    [5]             EF at %MW5.507
```

Set to display ASCII characters

ST

```
IF Enable THEN
      F95_ASC( s:= ’ABC1230 DEF’ ,
             d_Start=> ASCII_Output[0] );
END_IF;
```

F96_SRC

Table data search (16–bit search)

Steps	7

Description

Searches for the value that is the same as **s1** in the block of 16–bit areas specified by **s2** (starting area) through **s3** (ending area) if the trigger **EN** is in the ON–state.

When the search operation is performed, the searching results are stored as follows: the number of data that is the same as **s1** is transferred to special data register DT9037/DT90037. The position the data is first found in, counting from the starting 16–bit area, is transferred to special data register DT9038/DT90038. Be sure that **s2** ≤ **s3**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F96	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	16–bit area or equivalent constant to store the value searched for
s2	INT, WORD	starting 16–bit area of the block
s3	INT, WORD	ending 16–bit area of the block

The variables **s1**, **s2** and **s3** have to be of the same data type.

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1	x	x	x	x	x	x	x	x	x	x
s2, s3	–	x	x	x	x	x	x	x	x	–

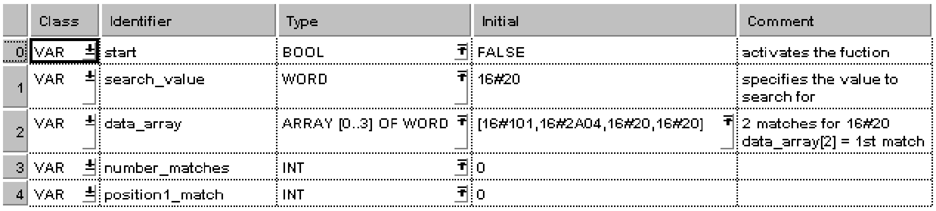
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

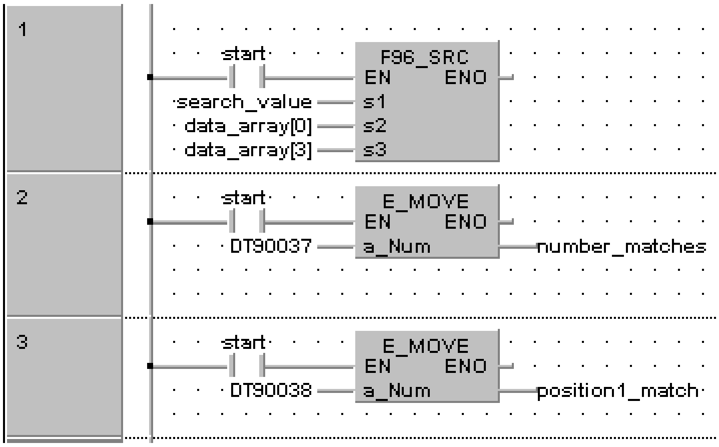
In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the fuction
1	VAR	search_value	WORD	16#20	specifies the value to search for
2	VAR	data_array	ARRAY [0..3] OF WORD	[16#101,16#2A04,16#20,16#20]	2 matches for 16#20 data_array[2] = 1st match
3	VAR	number_matches	INT	0	
4	VAR	position1_match	INT	0	

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F96_SRC( s1:= search_value ,
             s2_Start:= data_array[0] ,
             s3_End:= data_array[3] );
      number_matches:=DT90037;
      position_1match:=DT90038;
END_IF;
```

F138_HMSS

h:min:s → s conversion

Steps 7

Description

Converts the hours, minutes, and seconds data stored in the 32–bit area specified by **s** to seconds data if the trigger **EN** is in the ON–state. The converted seconds data is stored in the 32–bit area specified by **d**. All hours, minutes, and seconds data to convert and the converted seconds data is BCD. The max. data input value is 9,999 hours, 59 minutes and 59 seconds, which will be converted to 35,999,999 seconds in BCD format.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F138	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DWORD	source area for storing hours, minutes and seconds data
d	DWORD	destination area for storing converted seconds data

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	–
d	–	x	x	x	x	x	x	x	x	–

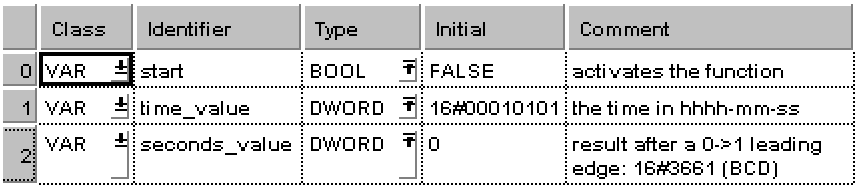
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

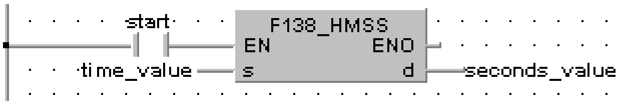


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	time_value	DWORD	16#00010101	the time in hhhh-mm-ss
2	VAR	seconds_value	DWORD	0	result after a 0->1 leading edge: 16#3661 (BCD)

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F138_HMSS(time_value, seconds_value);
END_IF;
```

F139_SHMS

s → h:min:s conversion

Steps	5

Description Converts the second data stored in the 32–bit area specified by **s** to hours, minutes, and seconds data if the trigger **EN** is in the ON–state. The converted hours, minutes, and seconds data is stored in the 32–bit area specified by **d**. The seconds prior to conversion and the hours, minutes, and seconds after conversion are all BCD data. The maximum data input value is 35,999,999 seconds, which is converted to 9,999 hours, 59 minutes and 59 seconds.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F139	x	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DWORD	source area for storing seconds data
d	DWORD	destination area for storing converted hours, minutes and seconds data

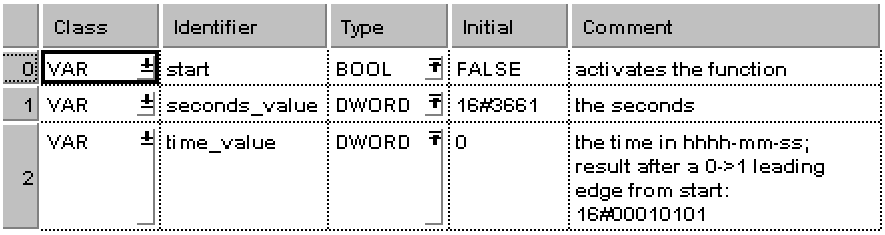
Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	–
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

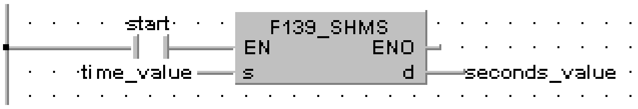
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	seconds_value	DWORD	16#3661	the seconds
2	VAR	time_value	DWORD	0	the time in hhhh-mm-ss; result after a 0->1 leading edge from start: 16#00010101

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F139_SHMS(seconds_value, time_value);
END_IF;
```

F327_INT

Floating point data → 16–bit integer data (the largest integer not exceeding the floating point data)

Steps	8

Description

The function converts a floating point data at input **s** in the range –32767.99 to 32767.99 into integer data (including +/– sign). The result of the function is returned at output **d**.
The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:
When there is a positive floating point value at the input, a positive pre–decimal value is returned at the output.
When there is a negative floating point value at the input, the next smallest pre–decimal value is returned at the output.
If the floating point value has only zeros after the decimal point, its pre–decimal point value is returned.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F327	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	REAL	source REAL number data (2 words)
d	INT	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	floating pt.
s	x	x	x	x	x	x	x	x	x	x
d	WX	WY	WR	WL	SV	EV	DT	LD	FL	floating pt.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

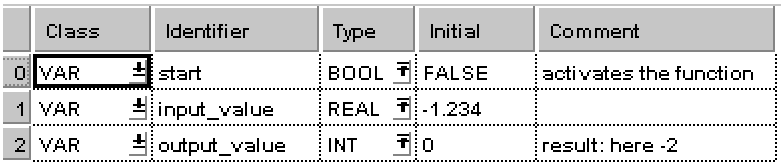
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	–the value at input s is not a REAL number, or the converted result exceeds the range of output d
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	– the result is 0

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



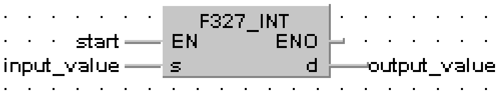
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-1.234	
2	VAR	output_value	INT	0	result: here -2

In this example, the input variable *input_value* is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable *start* is set to TRUE, the function is carried out. It converts the floating point value –1.234 into the whole number value –2, which is transferred to the variable *output_value* at the output. Since the whole number may not exceed the floating point value, the function rounds down here.

LD



ST

```
IF start THEN
      F327_INT(input_value, output_value);
END_IF;
```

F328_DINT

Floating point data → 32–bit integer data (the largest integer not exceeding the floating point data)

Steps	8

Description

The function converts a floating point data at input **s** in the range –2147483000 to 214783000 into integer data (including +/– sign). The result of the function is returned at output **d**.

The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:

When there is a positive floating point value at the input, a positive pre–decimal value is returned at the output.

When there is a negative floating point value at the input, the next smallest pre–decimal value is returned at the output.

If the floating point value has only zeros after the decimal point, its pre–decimal point value is returned.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F328	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	REAL	source REAL number data (2 words)
d	DINT	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	floating pt.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

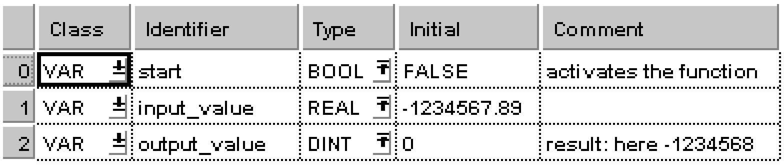
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	–the value at input **s** is not a REAL number, or the converted result exceeds the range of output **d**
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	– the result is 0

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



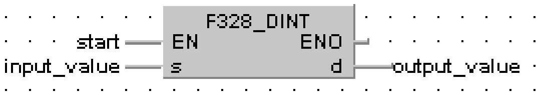
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	-1234567.89	
2	VAR	output_value	DINT	0	result: here -1234568

In this example, the input variable *input_value* is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable *start* is set to TRUE, the function is carried out. It converts the floating point value –1234567.89 into the whole number value –1234568, which is transferred to the variable *output_value* at the output. Since the whole number may not exceed the floating point value, the function rounds down here.

LD



ST

```
IF start THEN
      F328_DINT(input_value, output_value);
END_IF;
```

F333_FINT

Rounding the first decimal point down

Steps	8

Description The function rounds down the decimal part of the real number data and returns it at output **d**.

The converted whole–number value at output **d** is always less than or equal to the floating–point value at input **s:**
If a positive floating–point value is at the input, a positive pre–decimal point value is returned at the output.
If a negative floating–point value is at the input, the next smallest pre–decimal point value is returned at the output.
If the negative floating–point value has only zeros after the decimal point, its pre–decimal point position is returned.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F333	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	floating pt.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

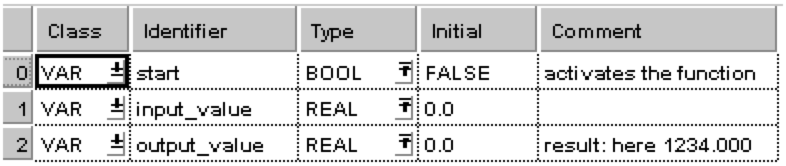
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	–the value at input s is not a REAL number
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	– the result is 0
R9009	%MX0.900.9	for an instant	–the result causes an overflow

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

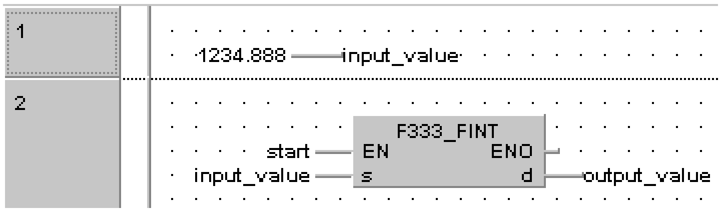


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	0.0	
2	VAR	output_value	REAL	0.0	result: here 1234.000

In this example, the input variable *input_value* is declared. However, you can write a constant directly at the input contact of the function instead.

Body The value 1234.888 is assigned to the variable *input_value*. When the variable *start* is set to TRUE, the function is carried out. It rounds down the *input_value* after the decimal point and returns the result (here: 1234.000) at the variable *output_value*.

LD



ST

```
input_value:=1234.888;
IF start THEN
      F333_FINT(input_value, output_value);
END_IF;
```

F334_FRINT

Rounding the first decimal point off

Steps 8

Description

The function rounds off the decimal part of the real number data and returns it at output **d**.

If the first post–decimal digit is between 0..4, the pre–decimal value is rounded down. If the first post–decimal digit is between 5..9, the pre–decimal value is rounded up.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F334	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	floating pt.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

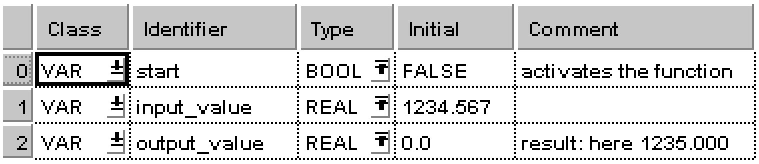
No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	–the value at input s is not a REAL number
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	– the result is 0
R9009	%MX0.900.9	for an instant	–the result causes an overflow

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



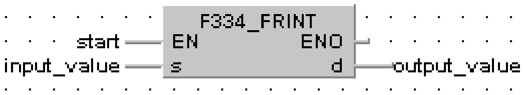
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	1234.567	
2	VAR	output_value	REAL	0.0	result: here 1235.000

In this example, the input variable *input_value* is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable *start* is set to TRUE, the function is carried out. It rounds off the *input_value* = 1234.567 after the decimal point and returns the result (here: 1235.000) at the variable *output_value*.

LD



ST

```
IF start THEN
      F334_FRINT(input_value, output_value);
END_IF;
```

F335_FSIGN Floating point data sign changes (negative/positive conversion)

Steps	8

Description The function changes the sign of the floating point value at input **s** and returns the result at output **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F335	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	REAL	source
d	REAL	destination

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	floating pt.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

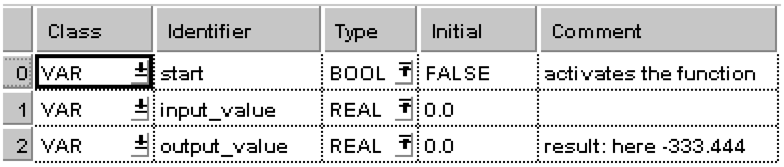
x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	–the value at input s is not a REAL number
R9008	%MX0.900.8	for an instant	
R9009	%MX0.900.9	for an instant	–the result causes an overflow

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header In the POU header, all input and output variables are declared that are used for programming this function.

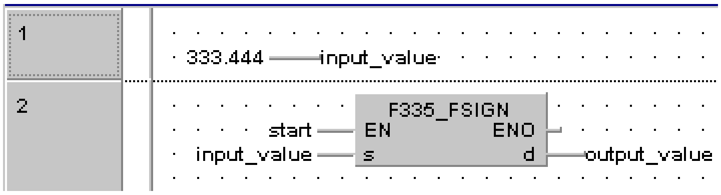


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	0.0	
2	VAR	output_value	REAL	0.0	result: here -333.444

In this example, the input variable *input_value* is declared. However, you can write a constant directly at the input contact of the function instead.

Body The value 333.4 is assigned to the variable *input_value*. When the variable *start* is set to TRUE, the function is carried out. The *output_value* is then –333.4.

LD



ST
```
input_value:=333.444;
IF start THEN
      F335_FSIGN(input_value, output_value);
END_IF;
```

F337_RAD

Conversion of angle units (Degrees → Radians)

Steps	8

Description The function converts the value of an angle entered at input **s** from degrees to radians and returns the result at output **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F337	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	REAL	source angle data (degrees), 2 words
d	REAL	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	floating pt.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

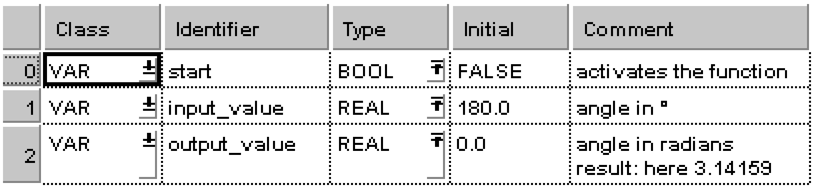
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	–the value at input s is not a REAL number
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	– the result is 0
R9009	%MX0.900.9	for an instant	–the result causes an overflow

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



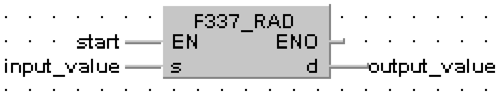
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	180.0	angle in °
2	VAR	output_value	REAL	0.0	angle in radians result: here 3.14159

In this example, the input variable *input_value* is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable *start* is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN
      F337_RAD(input_value, output_value);
END_IF;
```

F338_DEG

Conversion of angle units (Radians → Degrees)

Steps 8

Description

The function converts the value of an angle entered at input **s** from radians to degrees and returns the result at output **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F338	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	REAL	source angle data (radians), 2 words
d	REAL	destination for storing converted data

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	floating pt.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

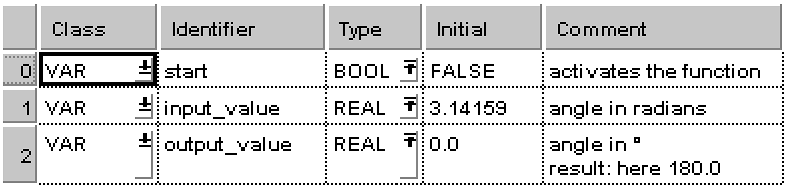
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	–the value at input s is not a REAL number
R9008	%MX0.900.8	for an instant	
R900B	%MX0.900.11	to TRUE	– the result is 0
R9009	%MX0.900.9	for an instant	–the result causes an overflow

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.



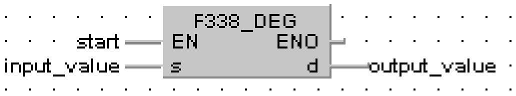
	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	input_value	REAL	3.14159	angle in radians
2	VAR	output_value	REAL	0.0	angle in ° result: here 180.0

In this example, the input variable *input_value* is declared. However, you can write a constant directly at the input contact of the function instead.

Body

When the variable *start* is set to TRUE, the function is carried out.

LD



ST

```
IF start THEN F338_DEG(input_value, output_value);
END_IF;
```

Chapter 21

Bit Manipulation Instructions

F130_BTS

16–bit data bit set

Steps	5

Description

Turns ON the bit specified by the bit position at **n** of the 16–bit data specified by **d** if the trigger **EN** is in the ON–state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F130	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	specifies bit position to be set

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

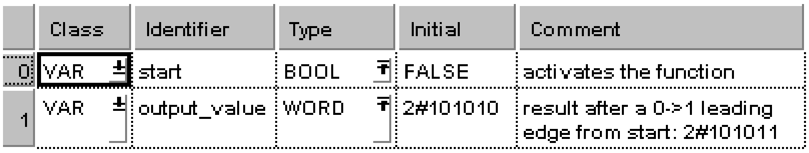
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

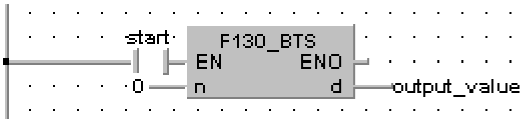


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#101010	result after a 0->1 leading edge from start: 2#101011

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F130_BTS( n:= 0,
             d=> output_value);
END_IF;
```

F131_BTR 16–bit data bit reset

Steps 5

Description

Turns OFF the bit specified by the bit position at **n** of the 16–bit data specified by **d** if the trigger **EN** is in the ON–state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F131	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	specifies bit position to be reset

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

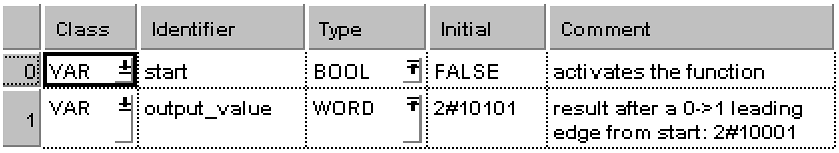
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

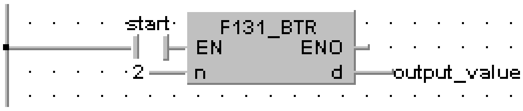


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#10101	result after a 0->1 leading edge from start: 2#10001

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F131_BTR( n:= 2,
             d=> output_value);
END_IF;
```

F132_BTI

16–bit data bit invert

Steps	5

Description Inverts [1 (ON) → 0 (OFF) or 0 (OFF) → 1 (ON)] the bit at bit position **n** in the 16–bit data area specified by **d** if the trigger **EN** is in the ON–state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F132	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	specify bit position to be inverted

Operands

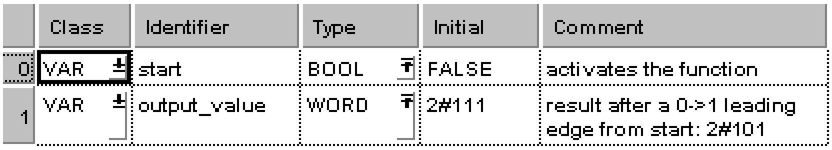
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

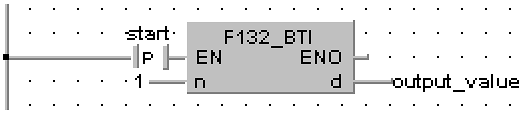


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	output_value	WORD	2#111	result after a 0->1 leading edge from start: 2#101

Body

When the variable *start* changes from FALSE to TRUE, the function is executed.

LD



ST

```
IF DF(start) THEN
      F132_BTI( n:= 1,
             d=> output_value);
END_IF;
```

F133_BTT

16–bit data test

Steps 5

Description

Checks the state [1 (ON) or 0 (OFF)] of bit position **n** in the 16–bit data specified by **d** if the trigger **EN** is in the ON–state. The specified bit is checked by special internal relay R900B.

- When specified bit is 0 (OFF), special internal relay R900B (=flag) turns ON.
- When specified bit is 1 (ON), special internal relay R900B (=flag) turns OFF.

n specifies the bit position to be checked in decimal data. Range of **n**: 0 to 15

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F133	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d	INT, WORD	16–bit area
n	INT	specifies bit position to be tested

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d	–	x	x	x	x	x	x	x	x	–
n	x	x	x	x	x	x	x	x	x	x

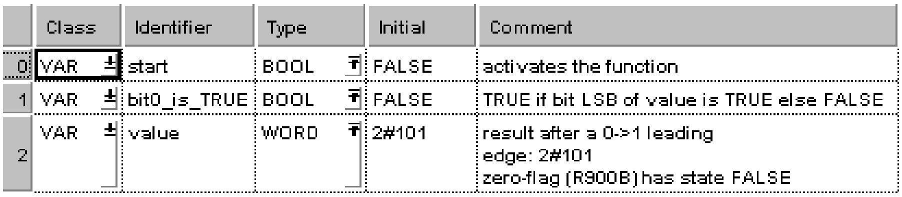
x: available
–: not available

Example

In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

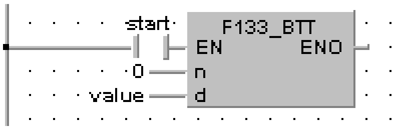


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	bit0_is_TRUE	BOOL	FALSE	TRUE if bit LSB of value is TRUE else FALSE
2	VAR	value	WORD	2#101	result after a 0->1 leading edge: 2#101 zero-flag (R900B) has state FALSE

Body

When the variable *start* is set to TRUE, the function is executed.

LD



```
ST    IF start THEN
            F133_BTT( n:= 0,
                   d:= value);
            IF R900B THEN
                  bit0_is_TRUE := FALSE;
            ELSE
                  bit0_is_TRUE := TRUE;
            END_IF;
      END_IF;
```

F135_BCU

Number of ON bits in 16–bit data

Steps	5

Description Counts the number of bits in the ON state (1) in the 16–bit data specified by **s** if the trigger **EN** is in the ON–state. The number of 1 (ON) bits is stored in the 16–bit area specified by **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F135	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	source
d	INT	destination area for storing the number of bits in the ON (1) state

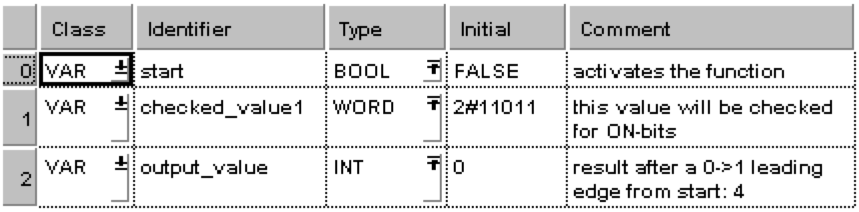
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

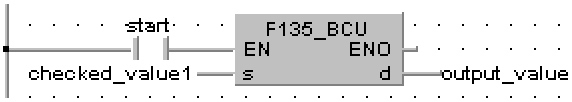
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	checked_value1	WORD	2#11011	this value will be checked for ON-bits
2	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 4

Body When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F135_BCU(checked_value1, output_value);
END_IF;
```

F136_DBCU

Number of ON bits in 32–bit data

Steps	7

Description Counts the number of bits in the ON state (1) in the 32–bit data specified by **s** if the trigger **EN** is in the ON–state. The number of 1 (ON) bits is stored in the 16–bit area specified by **d**.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F136	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DINT, DWORD	source
d	INT	destination area for storing the number of bits in the ON (1) state

Operands

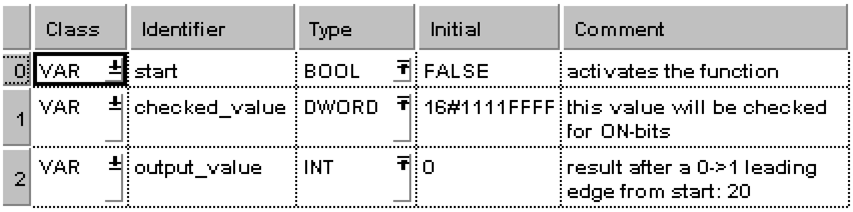
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

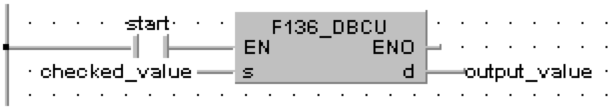


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function
1	VAR	checked_value	DWORD	16#1111FFFF	this value will be checked for ON-bits
2	VAR	output_value	INT	0	result after a 0->1 leading edge from start: 20

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F136_DBCU(checked_value, output_value);
END_IF;
```

Chapter 22

Process Control Instructions

F355_PID

PID processing instruction

Steps 4

☞ **We recommend using the Matsushita standard function blocks PID_FB or PID_FB_DUT. They are available for the FP–Sigma, FP0, FP2, FP2SH or FP10SH. They allow you to easily set parameters and correctly switch from manual to automatic operation. For details, see Online Help.**

Description

The PID processing instruction is used to regulate a process (e.g. a heater) given a measured value (e.g. temperature) and a predetermined output value (e.g. 20°C).
The function calculates a PID algorithm whose parameters are determined in a data table in the form of an ARRAY with 31 elements that is entered at input **s.** The data table contains the following parameters:
Control:]Control mode
SP:Set point value
PV:Process value
MV:Manipulated value
LowerLimit]:Output lower limit
UpperLimit]:Output upper limit
Kp:Proportional gain
Ti:Integral time
Td:Derivative time
Ts:Control cycle
AT_Progress]:Auto–tuning progress
ARRAY[11] through **ARRAY[30]**: are utilized internally by the PID controller.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F355	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	ARRAY [0..30] of INT or WORD	please see description section

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	–	–	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– the parameter settings are outside the allowed range
R9008	%MX0.900.8	for an instant	

■Detailed description of the data table for F355_PID

ARRAY[0]: Control mode

With this you select the type of PID processing and the activation (X = 8) of the auto–tuning.

16#X000:	Reverse operation PI–D
16#X001:	Forward operation PI–D
16#X002:	Reverse operation I–PD
16#X003:	Forward operation I–PD

☞ **The I–PD processing is somewhat more flexible than the PI–D processing and therefore needs more time to adjust.**

Forward and Reverse operation:
Reverse operation:The output value (MV) rises when the measured value (PV) sinks (e.g. heating).
Forward operation: The output value (MV) rises when the measured value (PV) rises (e.g. cooling).

– **Control: Auto–tuning**
When the most significant bit (MSB) in Control is set to 1, the auto tuning is activated. The optimum values for the PID parameters Kp, Ti, and Td are determined by measuring the responses of the process and are stored in Kp, Ti, and Td. Thereafter the auto tuning is deactivated (MSB in Control is set to 0). Since some operations do not permit auto tuning, the MSB in Control can be reset to 0 during the auto tuning process, thereby stopping the auto tuning. In this case the processing is carried out based on the original parameters. During auto–tuning is activ the output value MV is switched from upper limit to lower limit to avoid any damage of systems that have to use different limits or a reduced output span.

– **SP: Set value**
Here you set the target value that should be reached through the control process. It should fall within the range of the measured value. When using an analogue input, you can use a range between 0 and 10000.

– **PV: Measured value (PV)**
Here you enter the measured value that you want to be corrected via the operation. An analogue–digital converter is necessary for this. Adjust it so that the range of the measured value corresponds to that of the set value.

– **MV: Output value**
The output value (the result of the PID operation) is stored in this data word. When using an analogue output, the range lies between 0 and 4000 or between –2000 and +2000.

– **LowerLimit: Output lower limit**
Here you enter a lower limit of the output value between 0 and 10000. The value must be smaller than the output value’s upper limit.

– **UpperLimit: Output upper limit**
Here you enter an upper limit of the output value between 1 and 10000. The value must be larger than the output value’s lower limit.

– **Kp: Proportional gain**
In this data word, you write the parameter Kp. The stored value multiplied by 0.1 corresponds to the actual value of Kp. Values in the range of 1 to 9999 (0.1 to 999.9 in 0.1 steps) can be entered. If the auto tuning control is activated, this value is automatically adjusted and rewritten.

– **Ti: Integral time**
In this data word, you write the parameter Ti. The stored value multiplied by 0.1 corresponds to the actual value of Ti. Values in the range of 1 to 30000 (0.1 to 3000s in 0.1s steps) can be entered. If the auto tuning control is activated, this value is automatically adjusted and rewritten.

– **Td: Derivative time**
In this data word, you write the parameter Td. The stored value multiplied by 0.1 corresponds to the actual value of Td. Values in the range of 1 to 10000 (0.1 to 1000s in 0.1s steps) can be entered. If the auto tuning control is activated, this value is automatically adjusted and rewritten.

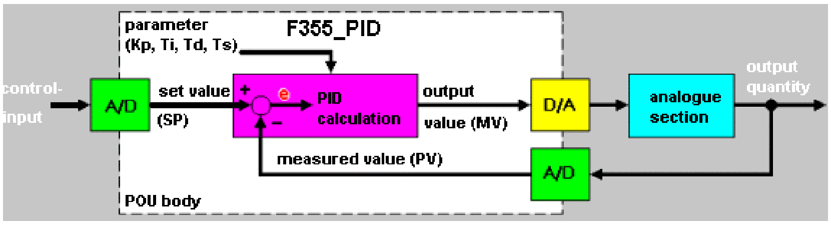
– **Ts: Control cycle**
Here you set the cycle for executing PID processing. The value of the data word multiplied by 0.01 corresponds to the actual value of Ts. Values in the range of 1 to 6000 (0.01s to 60.0s in 0.01s steps) can be entered.

– **AT_Progress:Auto–tuning progress**
When auto tuning is selected for the specified control mode (Control), a value from 1 to 5 will be stored indicating the progress of auto tuning.

– **ARRAY[11..30]: PID work area**
The function F355_PID uses this work area internally to calculate the PID operation.

Explanation of the operation of F355_PID

Standard structure of the controller loop with PID processing instruction.

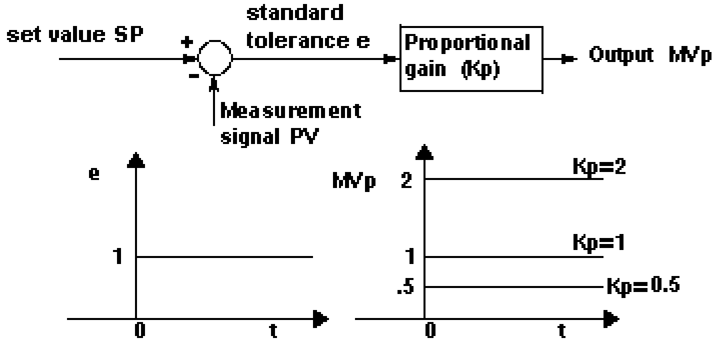


The above POU body represents the standard control loop.The control input is determined by the user (e.g. desired room temperature of 22°C). After the A/D conversion the set value (SP) is entered as the input value for the PID processing instruction.The measured value (PV) (e.g. current room temperature) is normally transmitted via a sensor and entered as the input value for the PID processor. F355_PID calculates the standard tolerance e from the set value and the measured value (e = set value – measured value). With the parameters given (proportional gain Kp, integral time Ti, ...) a new output value (MV) is calculated in increments set by the control cycle Ts. This result is then applied to the actuator (e.g. a fan that regulates room temperature) after the D/A conversion.The analogue section represents the system’s actuator, e.g. heater and temperature regulation of a room.

A PID operation consists of three components:

1. Proportional part (P part)

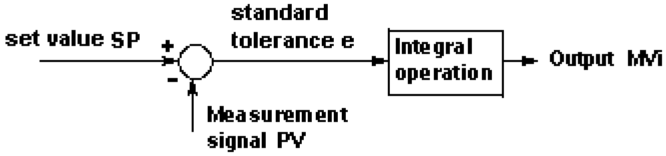
A proportional part generates an output that is proportional to the input. The proportional gain Kp determines by how much the input value is increased or decreased. A proportional part can be a simple electric resistor or a linear amplifier, for example.



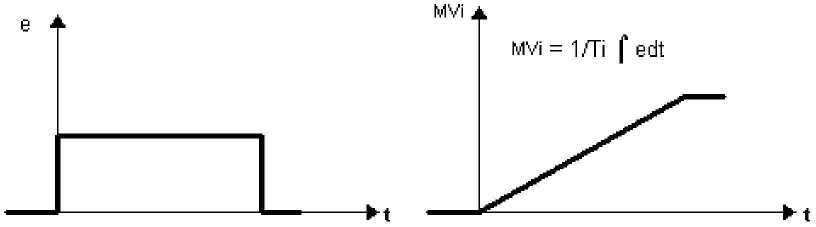
The P part displays a relatively large maximum overshot, a long settling time and a constant standard tolerance.

2. Integral part (I part)

An integral part produces an output quantity that corresponds to the time integral and input quantity (area of the input quantity). The integral time thus evaluates the output quantity **MVi**.
The integral part can be a quantity scale of a tank that is filled by a volume flow, for example. Because of the slow reaction time of the integral part, it has a larger maximum overshot than the P component, but no constant standard tolerance.



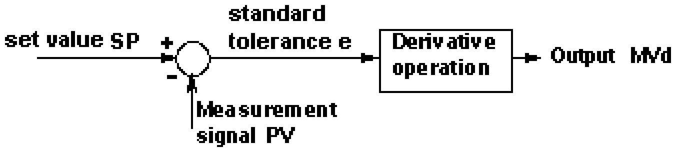
Example **Input quantity** e **and the output quantity** MVi **produced**



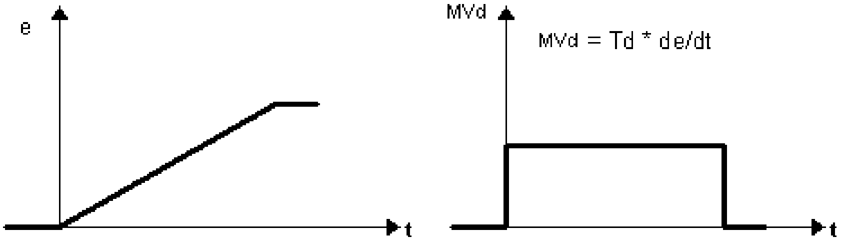
3. Derivative part (D part)

The derivative part produces an output quantity that corresponds to the time derivation of the input quantity. The derivative time corresponds to the weighting of the derived input quantity.

A derivative component can be an RC–bleeder (capacitor hooked up in series and resistance in parallel), for example.

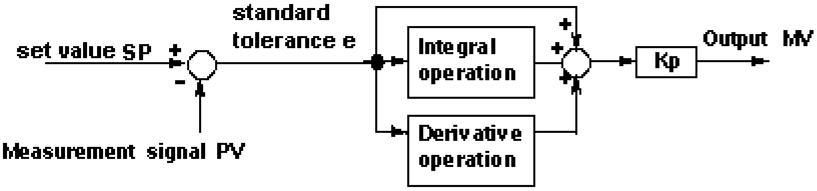


Example **Input quantity** e **and the output quantity** MVd **produced**



4. PID controller

A PID controller is a combination of a P component, an I component and a D component. When the parameters Kp, Ti and Td are optimally adjusted, a PID controller can quickly control and maintain a quantity at a predetermined set value.



Reference equations for calculating the controller output MV

The following equations are used to calculate the controller output MV under the following conditions:

In general:

The output value at time period **n** is calculated from the previous output value (n–1) and the change in the output value in this time interval.

$$MV_n = MV_{n-1} + \Delta MV$$

Reverse operation PI–D ARRAY[0] = 16#X000

$$\Delta MV = Kp \cdot [(e_n - e_{n-1}) + e_n \cdot Ts/Ti + \Delta D_n]$$

$$e_n = SP_n - PV_n$$

$$\Delta D_n = (\eta\beta - 1)D_{n-1} + \beta(PV_{n-1} - PV_n)$$

$$\eta = 1/8 \ \text{(constant)}$$

$$\beta = Td/(Ts + \eta Td)$$

Forward operation PI–D ARRAY[0] = 16#X001

$$\Delta MV = Kp \cdot [(e_n - e_{n-1}) + e_n \cdot Ts/Ti + \Delta D_n]$$

$$e_n = PV_n - SP_n$$
$$\Delta D_n = (\eta\beta - 1)D_{n-1} + \beta(PV_n - PV_{n-1})$$
$$\eta = 1/8 \ \ (\text{constant})$$
$$\beta = Td/(Ts + \eta Td)$$

Reverse operation I–PD ARRAY[0] = 16#X002

$$\Delta MV = Kp \cdot [(PV_{n-1} - PV_n) + e_n \cdot Ts/Ti + \Delta D_n]$$

$$e_n = SP_n - PV_n$$
$$\Delta D_n = (\eta\beta - 1)D_{n-1} + \beta(PV_{n-1} - PV_n)$$
$$\eta = 1/8 \ \ (\text{constant})$$
$$\beta = Td/(Ts + \eta Td)$$

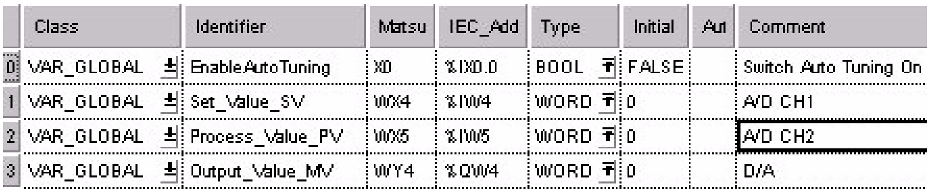
Forward operation I–PD ARRAY[0] = 16#X003

$$\Delta MV = Kp \cdot [(PV_n - PV_{n-1}) + e_n \cdot Ts/Ti + \Delta D_n]$$

$$e_n = PV_n - SP_n$$
$$\Delta D_n = (\eta\beta - 1)D_{n-1} + \beta(PV_n - PV_{n-1})$$
$$\eta = 1/8 \ \ (\text{constant})$$
$$\beta = Td/(Ts + \eta Td)$$

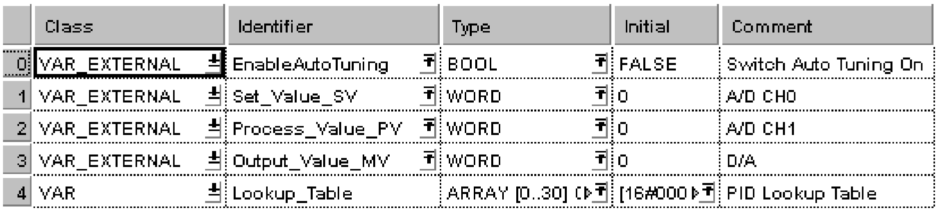
Example In this example the function F355_PID is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

GVL In the **G**lobal **V**ariable **L**ist, you define variables that can be accessed by all POUs in the project.



	Class	Identifier	Matsu	IEC_Add	Type	Initial	Au	Comment
0	VAR_GLOBAL	EnableAutoTuning	X0	%IX0.0	BOOL	FALSE		Switch Auto Tuning On
1	VAR_GLOBAL	Set_Value_SV	WX4	%IW4	WORD	0		A/D CH1
2	VAR_GLOBAL	Process_Value_PV	WX5	%IW5	WORD	0		A/D CH2
3	VAR_GLOBAL	Output_Value_MV	WY4	%QW4	WORD	0		D/A

POU header In the POU header, all input and output variables are declared that are used for programming this function.



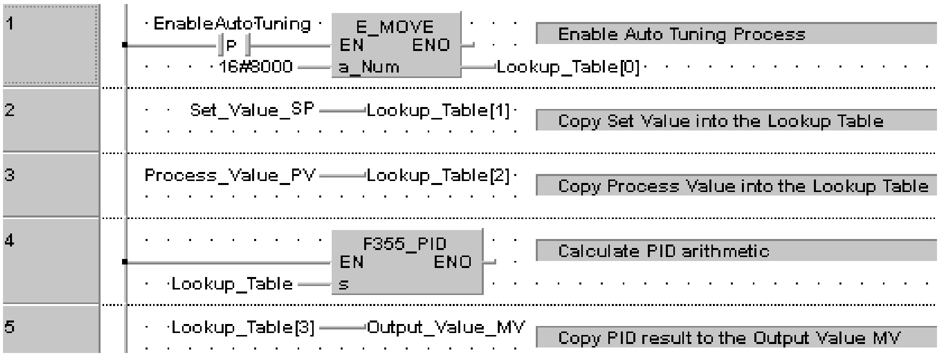
	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	EnableAutoTuning	BOOL	FALSE	Switch Auto Tuning On
1	VAR_EXTERNAL	Set_Value_SV	WORD	0	A/D CH0
2	VAR_EXTERNAL	Process_Value_PV	WORD	0	A/D CH1
3	VAR_EXTERNAL	Output_Value_MV	WORD	0	D/A
4	VAR	Lookup_Table	ARRAY [0..30] O	[16#000	PID Lookup Table

In the initialization of the ARRAY *Lookup_Table*, the upper limit of the controller output is set to 4000. The proportional gain Kp is initially set at 80 (8), Ti and Td at 200 (20s) and the control cycle Ts at 100 (1s).

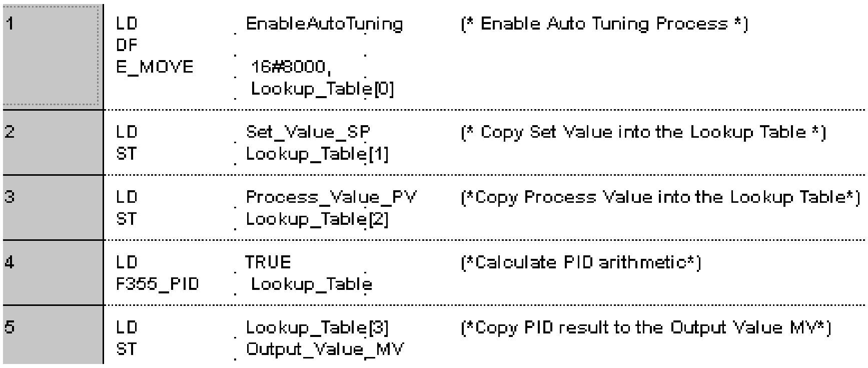
Body

The standard function E_MOVE copies the value 16#8000 to the first element of the *Lookup_Table* when the variable activeautotuning is set from FALSE to TRUE (i.e. activates the control mode auto tuning in the function F355_PID). The variables *Set_Value_SP* and *Process_Value_PV* are assigned to the second and third elements of data table. They receive their values from the A/D converter CH0 and CH1. Because of EN input of F355_PID is connected to the power rail, the function is carried out, when the PLC is in RUN mode. The calculated controller output is stored in the fourth element of data table and assigned to the variable *Output_Value_MV*. Its value is returned via a D/A converter from the PLC to the output of the system.

LD



IL



```
1   LD        EnableAutoTuning     (* Enable Auto Tuning Process *)
    DF
    E_MOVE     16#8000,
               Lookup_Table[0]

2   LD        Set_Value_SP         (* Copy Set Value into the Lookup Table *)
    ST        Lookup_Table[1]

3   LD        Process_Value_PV     (*Copy Process Value into the Lookup Table*)
    ST        Lookup_Table[2]

4   LD        TRUE                 (*Calculate PID arithmetic*)
    F355_PID   Lookup_Table

5   LD        Lookup_Table[3]      (*Copy PID result to the Output Value MV*)
    ST        Output_Value_MV
```

Chapter 23

Timer Instructions

TM_1s

Timer for 1s intervals

Steps 4–5

Description

The **TM_1s** instruction sets the ON–delay timer for 1s units (0 to 32767s).

The areas used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON–state, the Preset (Set) value is transferred to the **EV** from the **SV**. During the timing operation, the time is subtracted from the **EV**. The scan time is also subtracted from the **EV** in the next scan. The timer contact **T** turns ON, when the **EV** becomes 0.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_1s	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	INT, WORD	timer address in system registers 5 and 6
SV	INT, WORD	set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
start	x	x	x	x	x	x	–	–	–	–
T	–	x	x	x	–	–	–	–	–	–
Num*	–	–	–	–	–	–	–	–	–	x
SV	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	–	–	–	x	–	–	–	–	x

x: available
–: not available

☞

- **It is not possible to use this function in a function block POU.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6. Timers of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

Example Below is an example of an instruction list (IL) body for the instruction.

```
LD      start   (* EN = start; Starting signal for
                the TM_1s function. *)
TM_1s   13,SV1  (* Num* = 13 (Address of the timer)
        3       *)
                (* SV = SV13 (containing the time
                for the timer) *)
ST      Var_0   (* T = Var_0; The variable Var_0
                turns ON, *)
                (*                    when the EV
                becomes 0. *)
```

TM_100ms

Timer for 100ms intervals

Steps 3–4

Description

The **TM_100ms** instruction sets the ON–delay timer for 0.1s units (0 to 3276.7s). The **TM** instruction is a down type preset timer.

The areas used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON–state, the Preset (Set) value is transferred to the **EV** from the **SV**. During the timing operation, the time is subtracted from the **EV**. The scan time is also subtracted from the **EV** in the next scan. The timer contact **T** turns ON, when the **EV** becomes 0.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_100ms	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	INT, WORD	timer address in system registers 5 and 6
SV	INT, WORD	set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
start	x	x	x	x	x	x	–	–	–	–
T	–	x	x	x	–	–	–	–	–	–
Num*	–	–	–	–	–	–	–	–	–	x
SV	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	–	–	–	x	–	–	–	–	x

x: available
–: not available

☞

- **It is not possible to use this function in a function block POU.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6. Timers of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In or-**

der to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.

Example Below is an example of an instruction list (IL) body for the instruction.

```
LD       start    (* EN = start; Starting signal for
                  the TM_100ms function. *)
         16,32123 (* Num* = 16 (Address of the timer)
TM_100ms          *)
                  (* SV = 32123 (Time, corresponding
                  3212,3 sec. ) *)
ST       Var_0    (* T = Var_0; The variable Var_0
                  turns ON, *)
                  (* when the EV becomes 0. *)
```

TM_10ms

Timer for 10ms intervals

Steps	3–4

Description The **TM_10ms** instruction sets the ON–delay timer for 0.01s units (0 to 327.67s).

The areas used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: EV

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON–state, the Preset (Set) value is transferred to the EV from the **SV**. During the timing operation, the time is subtracted from the EV. The scan time is also subtracted from the EV in the next scan. The timer contact **T** turns ON, when the EV becomes 0.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_10ms	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
start	BOOL	starts timer
Num*	INT, WORD	timer address in system registers 5 and 6
SV	INT, WORD	set value
T	BOOL	timer contact

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
start	x	x	x	x	x	x	–	–	–	–
T	–	x	x	x	–	–	–	–	–	–
Num*	–	–	–	–	–	–	–	–	–	x
SV	**WX**	**WY**	**WR**	**WL**	**SV**	**EV**	**DT**	**LD**	**FL**	**dec. or hex.**
	–	–	–	–	x	–	–	–	–	x

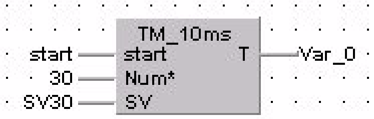
x: available
–: not available

☞

- **It is not possible to use this function in a function block POU.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6. Timers of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In or-**

der to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.

Example Below is an example of a ladder diagram (LD) body for the instruction.



TM_1ms

Timer for 1ms intervals

Steps 3–4

Description

The **TM_1ms** instruction sets the ON–delay timer for 0.001s units (0 to 32.767s).

The areas used for the instruction are:

- Preset (Set) value area: **SV**
- Count (Elapsed) value area: **EV**

When the mode is set to RUN mode, the Preset (Set) value is transferred to the **SV**. If the trigger of the timer instruction **start** is in the ON–state, the Preset (Set) value is transferred to the **EV** from the **SV**. During the timing operation, the time is subtracted from the **EV**. The scan time is also subtracted from the **EV** in the next scan. The timer contact **T** turns ON, when the **EV** becomes 0.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
TM_1ms	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
start	BOOL	starts timer
T	BOOL	timer contact
Num*	INT, WORD	timer address in system registers 5 and 6
SV	INT, WORD	set value

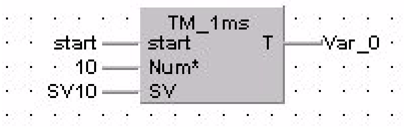
Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
start	x	x	x	x	x	x	–	–	–	–
T	–	x	x	x	–	–	–	–	–	–
Num*	–	–	–	–	–	–	–	–	–	x
SV	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	–	–	–	x	–	–	–	–	x

x: available
–: not available

- **It is not possible to use this function in a function block POU.**
- **Every used timer must have a separate constant Num*. Available Num* addresses depend on system registers 5 and 6. Timers of type TM_1s, TM_100ms, TM_10ms, TM_1ms use the same Num* address range.**
- **The Matsushita timer functions (TM_1s, TM_100ms, TM_10ms, and TM_1s) use the same NUM* address area as the Matsushita timer function blocks (TM_1s_FB, TM_100ms_FB, TM_10ms_FB, and TM_1s_FB). For the timer function blocks the compiler automatically assigns a NUM* address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address. In order to avoid errors (address conflicts), these timer functions and function blocks should not be used together in a project.**

Example Below is an example of a ladder diagram (LD) body for the instruction.



F137_STMR

Auxiliary timer (sets the ON–delay timer for 0.01s units)

Steps	5

Description

The auxiliary timer instruction **F137 (STMR)** is a down type timer. The formula of the timer–set time is 0.01 sec. * set value **s** (time can be set from 0.01 to 327.67 sec.). If you use the special internal relay R900D as the timer contact, be sure to program it at the address immediately after the instruction.
Timer operation:

- If the trigger **EN** of the auxiliary timer instruction (STMR) is in the ON–state, the constant or value specified by **s** is transferred to the area specified by **d**.
- During the timing operation, the time is subtracted from the value in the area specified by **d**.
- The output ENO turns ON when the value in the area specified by **d** becomes 0.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F137	x	–	5k	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	16–bit area or equivalent constant for timer set value
d	INT, WORD	16–bit area for timer elapsed value

The variables **s** and **d** have to be the same data type.

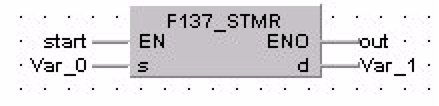
Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example

Below is an example of a ladder diagram (LD) body for the instruction.



F183_DSTM Special 32–bit timer

Steps	7

Description The F183 instruction activates an upward counting 32–bit timer which works on–delayed. The smallest counting unit is 0.01s. During execution of F183 (start = TRUE), elapsing time is added to the elapsed value **d**. The timer output will be enabled when the elapsed value **d** equals the set value **s**. If the start condition start is set to FALSE, execution will be interrupted and the elapsed value **d** will be reset to zero. The set value **s** can be changed during execution of F183.

The delay time of the timer can be calculated using the following formula: (Set Value s) * (0.01s) = on–delay

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F183	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	DINT, DWORD	set value, range 0 to 2147483647
d	DINT, DWORD	elapsed value, range 0 to 2147483647

Operands

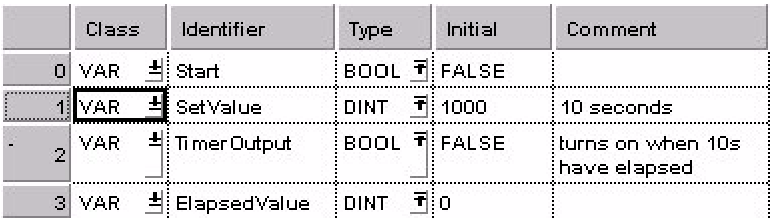
For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	–	x	x	x	–	–	x
d	–	x	x	–	x	x	x	–	–	–

x: available
–: not available

Example In this example the function F183_DSTM is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

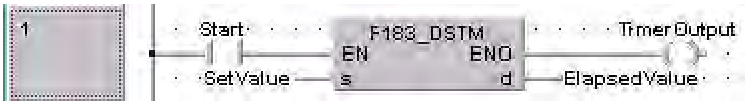


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR	SetValue	DINT	1000	10 seconds
2	VAR	TimerOutput	BOOL	FALSE	turns on when 10s have elapsed
3	VAR	ElapsedValue	DINT	0	

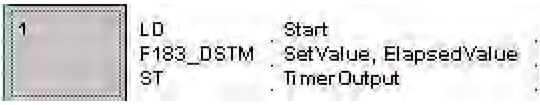
Body

This example uses a variable at the input contact. You may also use a constant.

LD



IL



```
1    LD          Start                   ;
     F183_DSTM   SetValue, ElapsedValue  ;
     ST          TimerOutput             ;
```

Chapter 24

Counter Instructions

CT Counter

Steps 3–4

Description

Decrements a preset counter. The function has the following parameters: **Count, Reset, Num*, SV,** and **C**. Their functions are listed in the Data types table below.

When the **Reset** input is on, the set value (**SV**) is reset to the value assigned to it. The set value can be set to a decimal constant from 0 to 32767.

When the **Count** input changes from off to on, the set value begins to decrement. When this value reaches 0, the counter output (**C**) turns on.

If the **Count** input and **Reset** input both turn on at the same time, the **Reset** input is given priority.

If the **Count** input rises and the **Reset** input falls at the same time, the count input is ignored and preset is executed.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
CT	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
Count	BOOL	subtracts 1 from the set value each time it is activated
Reset	BOOL	resets the counter when it is ON
Num*	decimal constant	number assigned to the counter (see System Register 5)
SV	INT, WORD	set value is the number the counter starts subtracting from
C	BOOL	the counter turns on when it reaches the SV

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
Count	x	x	x	x	x	x	–	–	–	–
Reset	x	x	x	x	x	x	–	–	–	–
Num*	–	–	–	–	–	–	–	–	–	x
C	–	x	x	x	–	–	–	–	–	–
SV	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	–	–	–	–	x	–	–	–	–	x

x: available
–: not available

Details about points

For FP–M/FP0 T32C/FP1, the following point numbers can be used for counters.

Type	Number of points	Nos. that can be used
FP–M C16T **FP1 C14, C16**	28 points	100 to 127
FP–M C20, C32 **FP0 T32C** **FP1 C24, C40, C56, C72**	44 points	100 to 143

The number of counter points can be changed using System Register 5. The number of points can be increased up to 3,072 with the FP2SH and FP10SH, up to 256 with the FP–C and FP3, up to 1,024 with the FP–Sigma and FP2, up to 128 with the FP–M C16T and FP1 C14, C16, and up to 144 with the FP–M C20, C32 and FP1 C24, C40, C56 and C72, and FP0. Be aware that increasing the number of counter points decreases the number of timer points.

The following point numbers can be used for counter depending on the type of FP0 C10/C14/C16/C32.

Type	Useable counter point numbers
FP0 C10, C14 and C16	44 points (C100 to C143) Non–hold type: 40 points (C100 to C139) Hold type: 4 points (C140 to C143)
FP0 C32	44 points (C100 to C143) Non–hold type: 28 points (C100 to C127) Hold type: 16 points (C128 to C143)

For all models except for the FP0 C10, C14, C16 and C32, there is a hold type, in which the counter status is retained even if the power supply is turned off, or if the mode is switched from RUN to PROG, and a non–hold type, in which the counter is reset under these conditions. System register 6 can be used to specify a non–hold type.

■ Set Value and Elapsed Value area

At the fall time when the reset input goes from on to off, the value of the set value area (SV) is preset in the elapsed value area (EV).

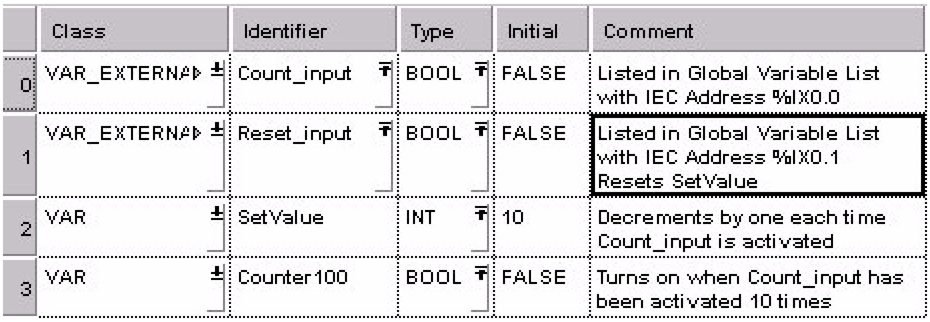
When the reset input is on, the elapsed value is reset to 0.

When the count input changes from off to on, the set value begins to decrement, and when the elapsed value reaches 0, the counter contact Cn (n is the counter number) turns on.

Example In this example the function CT is programmed in ladder diagram (LD).

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

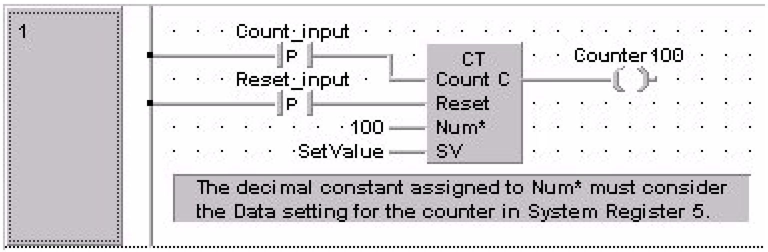


	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNA	Count_input	BOOL	FALSE	Listed in Global Variable List with IEC Address %IX0.0
1	VAR_EXTERNA	Reset_input	BOOL	FALSE	Listed in Global Variable List with IEC Address %IX0.1 Resets SetValue
2	VAR	SetValue	INT	10	Decrements by one each time Count_input is activated
3	VAR	Counter100	BOOL	FALSE	Turns on when Count_input has been activated 10 times

Body

The set value SV is set to 10 when *Reset_input* is activated. Each time *Count_input* is activated, the value of SV decreases by 1. When this value reaches 0, *Counter100* turns on. *Num** is assigned the counter number, which must be equal to or greater to the number assigned to Data in System Register 5.

LD



The decimal constant assigned to Num* must consider the Data setting for the counter in System Register 5.

F118_UDC

UP/DOWN counter

Steps 5

Description

UD_Trig: DOWN counting if the trigger is in the OFF state.
UP counting if the trigger is in the ON state.

Cnt_Trig: Adds or subtracts one count at the leading edge of this trigger.

Rst_Trig: The condition is reset when this signal is on.

The area for the elapsed value **d** becomes 0 when the leading edge of the trigger is detected (OFF → ON). The value in **s** is transferred to **d** when the trailing edge of the trigger is detected (ON → OFF).

s: Preset (Set) value or area for Preset (Set) value.

d: Area for count (elapsed) value.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
CT	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
UD_Trig	BOOL	sets counter to count up (ON) or down (0FF)
Cnt_Trig	BOOL	starts counter
Rst_Trig	BOOL	resets counter
s	INT, WORD	16–bit area or equivalent constant for counter preset value
d	INT, WORD	16–bit area for counter elapsed value

The variables **s** and **d** have to be of the same data type.

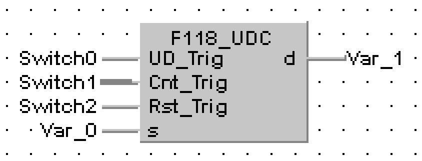
Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
UD_Trig, Cnt_Trig, Rst_Trig	x	x	x	x	x	x	–	–	–	–
s	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Example

Below is an example of a ladder diagram (LD) body for the instruction.



Chapter 25

High–Speed Counter Special Instructions

F0_MV

High–speed counter control

Steps 5

Description Controls the software reset, disabling of the counter, and stops pulse outputs.

For more information on the high–speed counter, pulse and PWM output, see Appendix A.

Description for FP0:

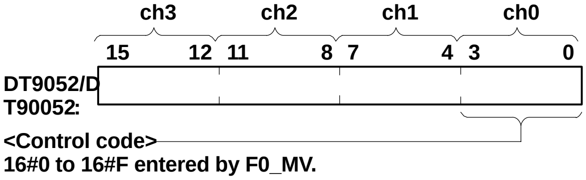
This instruction is used when performing the following operations while using the high–speed counter:

- Performing a software reset.
- Disabling the count.
- Temporarily disabling the hardware reset by the external input X2 and X5.
- Stopping the positioning and pulse outputs.
- Clearing the controls executed with the high–speed counter instructions F166, F167, F168, F169, and F170.
- Setting the near home input during home return operations for decelerating the speed of movement.
- When a control code is programmed once, it is saved until the next time it is programmed.

High–speed counter control register (DT9052/DT90052)

The control code program area DT9052/DT90052 divides 4 bits to each channel of the high–speed counter.

The control code entered in the **F0_MV** instruction is stored in special data register DT9052/DT90052.



Precautions during programming for FP0:

- **The hardware reset disable is only effective when using reset inputs (X2 and X5).**
- **Count disable and software reset during home return operations does not allow near home processing.**

- **The near home bit is saved, however, to cause near home processing during home return operations, it is necessary to enter 1 to the corresponding bit each cycle.**

■ Description for FP–M/FP1:

- Performing a software reset.
- Disabling the count.
- Temporarily disabling the hardware reset by the external input X2.
- Stopping the pulse outputs.
- Resetting and turning off the pattern output and cam output.
- Clearing the controls executed with the F162_HC0S, F163_HC0R, F164_SPD0 and F165_CAM0 instructions.

Special data register DT9052 stores control code and high–speed counter modes as follows:

Bit position	15 . . 12	11 . . 8	7 . . 4	3 . . 0
DT9052				

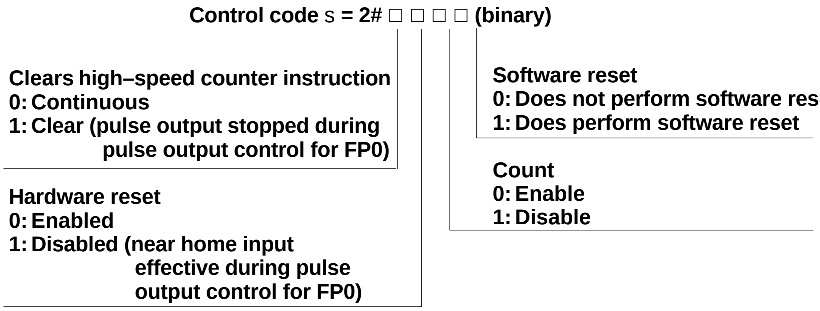
High–speed counter modes specified in system register 400 (16#0 to 16#8)

Control code 16#0 to 16#F entered by F0_MV instruction.

Precautions during programming for FP–M/FP1:

- **The outputs used for the F164_SPD0, and F165_CAM0 instructions are turned off.**
- **Special internal relays R903A (high–speed counter control flag) and R903B (cam control flag) turn off and the elapsed value is not clear while 1 is set to bit position 3 of DT9052.**
- **The control operations of the high–speed counter instructions “F162_HC0S, F163_HC0R, F164_SPD0, and F165_CAM0” are executed continuously when 0 is set to bit position 3 of DT9052.**

Specifying the control code “s”



Control code s **= 2# □ □ □ □ (binary)**

Clears high–speed counter instruction
0: Continuous
1: Clear (pulse output stopped during pulse output control for FP0)

Hardware reset
0: Enabled
1: Disabled (near home input effective during pulse output control for FP0)

Software reset
0: Does not perform software reset
1: Does perform software reset

Count
0: Enable
1: Disable

e.g.16#1, perform software reset
16#2, count disable
16#4, hardware reset disable
16#8, clear high–speed counter instruction

Data types

Variable	Data type	Function
s	INT, WORD	specifies high–speed counter operation
d	INT, WORD	controls high–speed counter operation at specified address, DT9052 (DT90052 for FP0 T32–CP)

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	x
d	–	x	x	x	x	x	x	x	x	–

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	the value of s exceeds the limit of specified range.
R9008	%MX0.900.8	for an instant	

Example

The following provides generic examples and explanations of F0_MV when used to control high–speed counter functions.

- Perform software reset . . . . 16#1(0001)
- Count disable . . . . 16#2(0010)
- Stop pulse output . . . . 16#8(1000)
- Turn off pulse output and reset elapsed value . 16#9(1001)

Enter the control code into the area DT9052/DT90052 of the corresponding channel.

For FP–M/FP1, when the mode is changed from PROG. to RUN, the lower–byte of DT9052 is set to 16#0.

16#0 (0000):

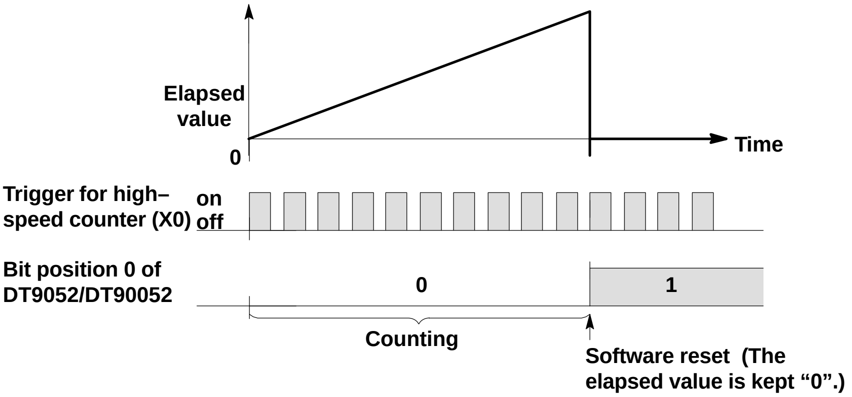
- Software reset operation is not performed.
- Count inputs are accepted.
- Reset input X2 enabled.
- The **F162_HC0S**, **F163_HC0R**, **F164_SPD0**, and **F165_CAM0** instructions continue to operate.

■ Operations of control code

① Software reset operation (bit position 0 of DT9052/DT90052)

When 0 is set to bit position 0 of DT9052/DT90052, the elapsed value counts continuously.

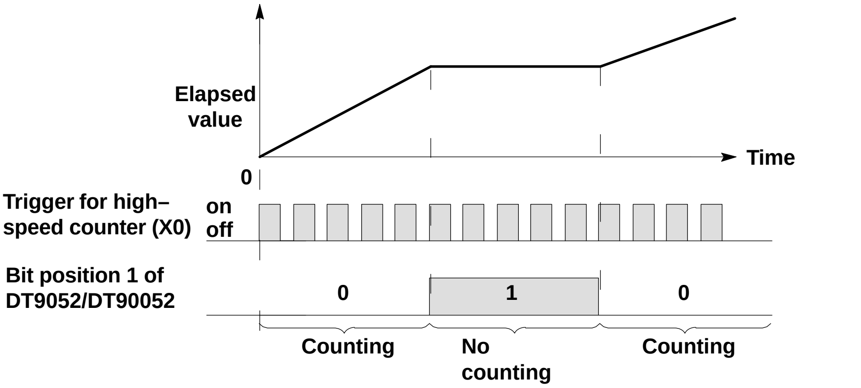
When 1 is set to bit position 0 of DT9052/DT90052, the elapsed value of the high–speed counter becomes 0 and keeps its value.


Elapsed value
0
Time
Trigger for high–speed counter (X0)
on
off
Bit position 0 of DT9052/DT90052
0
1
Counting
Software reset (The elapsed value is kept “0”.)

② Count input control operation (bit position 1 of DT9052/DT90052)

When 0 is set to bit position 1 of DT9052/DT90052, the count input is enabled

While 1 is set to bit position 1 of DT9052/DT90052, the count input is disabled (no counting) and the current elapsed value is kept.

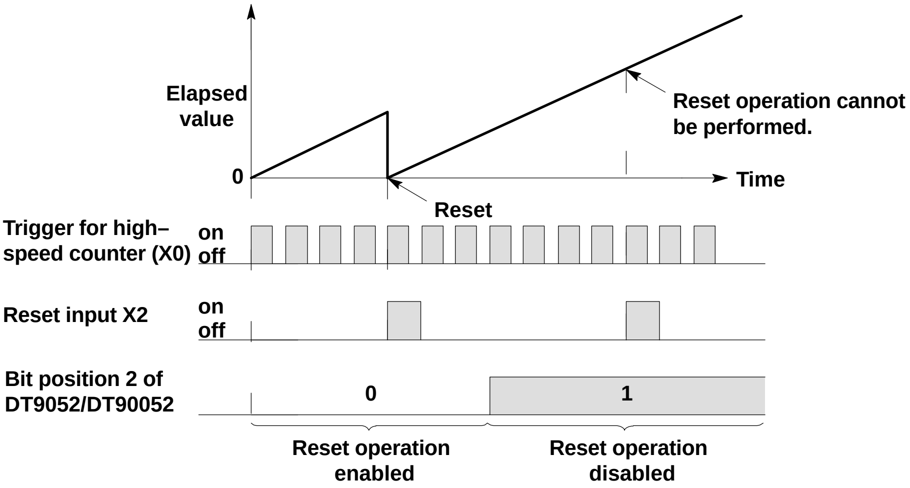

Elapsed value
0
Time
Trigger for high–speed counter (X0)
on
off
Bit position 1 of DT9052/DT90052
0
1
0
Counting
No counting
Counting

③ Hardware reset control operation (bit position 2 of DT9052/DT90052)

Even if reset input X2 is turned on, the reset operation cannot be performed when 1 is set to bit position 2 of DT9052/DT90052.

The hardware reset input is enabled when 0 is set to bit position 2 of DT9052/DT90052.

You can use reset operation only after system register 400 is set using X2 as the reset input (set value is even number: 16#2, 16#4, 16#6, or 16#8).

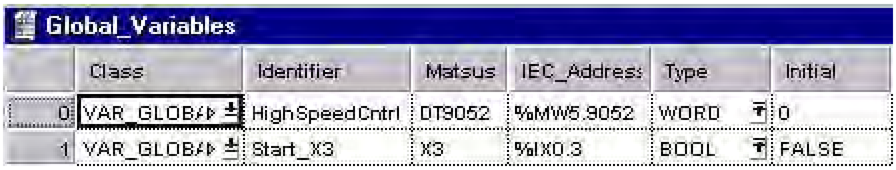

Elapsed value
0
Time
Reset operation cannot be performed.
Reset
Trigger for high–speed counter (X0)
on
off
Reset input X2
on
off
Bit position 2 of DT9052/DT90052
0
1
Reset operation enabled
Reset operation disabled

④ Control of high–speed counter instructions (bit position 3 of DT9052/DT90052)

The control operations of the **F162_HC0S**, **F163_HC0R**, **F164_SPD0**, and **F165_CAM0** instructions are stopped and cleared when 1 is set to bit position 3 of DT9052/DT90052.

Example In this example the function F0_MV is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

GVL In the **G**lobal **V**ariable **L**ist, you define variables that can be accessed by all POUs in the project.

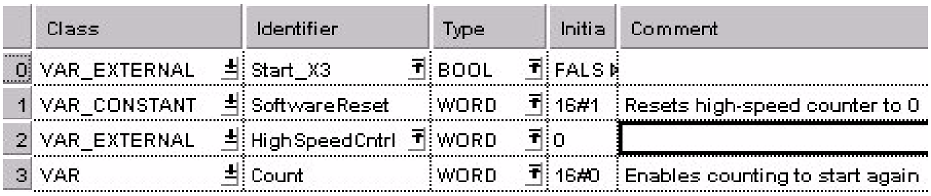


Global_Variables

	Class	Identifier	Matsus	IEC_Address	Type	Initial
0	VAR_GLOBAL	HighSpeedCntrl	DT9052	%MW5.9052	WORD	0
1	VAR_GLOBAL	Start_X3	X3	%IX0.3	BOOL	FALSE

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

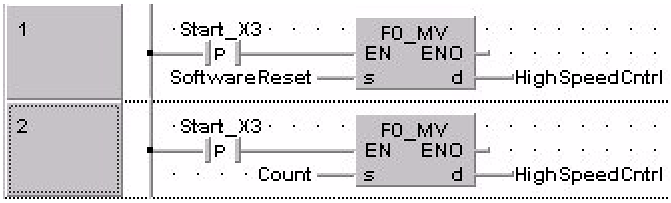


	Class	Identifier	Type	Initia	Comment
0	VAR_EXTERNAL	Start_X3	BOOL	FALS	
1	VAR_CONSTANT	SoftwareReset	WORD	16#1	Resets high-speed counter to 0
2	VAR_EXTERNAL	HighSpeedCntrl	WORD	0	
3	VAR	Count	WORD	16#0	Enables counting to start again

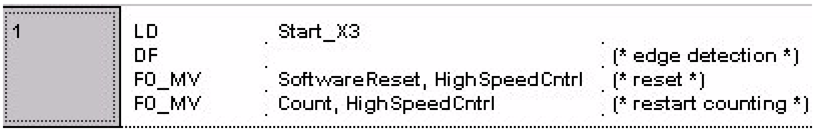
Body

The elapsed value of the high–speed counter is reset to zero (16#1) the first time F0_MV is executed and counting begins again (16#0) the second time it is executed.

LD



IL



```
1   LD       Start_X3
    DF                                          (* edge detection *)
    F0_MV    SoftwareReset, HighSpeedCntrl      (* reset *)
    F0_MV    Count, HighSpeedCntrl              (* restart counting *)
```

F162_HC0S High–speed counter output set

Steps	7

Description Sets the value specified by s as target value of the high–speed counter if the trigger **EN** is in the ON–state. When the elapsed value (DT9045 and DT9044) of the high–speed counter agrees with the target value (DT9047 and DT9046), the external output relay specified by **d** turns ON. You can use 8 external output relays (Y0 to Y7). The target value is stored in special data registers DT9047 and DT9046 when the F162 (HC0S) instruction is executed and it is cleared when the elapsed value of the high–speed counter agrees with the target value. Use 24 bit binary data with sign data for the target value of HSC (FF800000 hex to 007FFFFF hex / –8,388,608 to 8,388,607). Special internal relay R903A turns ON and stays ON while the F162 (HC0S) instruction is executed and it is cleared when the elapsed value of the high–speed counter coincides with the target value. Even if the reset operation of the high–speed counter is performed after executing the F162 (HC0S) instruction, the target value setting is not cleared until the elapsed value of the high–speed counter coincides with the target value. To reset the external output relay, which is set ON by the F162 (HC0S) instruction, use the F163_HC0R instruction. You can use the same external output relay specified by the F162 (HC0S) instruction in other parts of program. It is not regarded duplicate use of the same output. While special internal relay R903A is in ON state, no other high–speed counter instructions F162 (HC0S), F163_HC0R, F164_SPD0, F165_CAM0 can be executed.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F162	–	x	x	x	x

x: available
–: not available

Data types

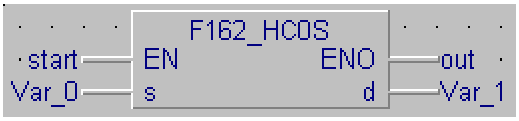
Variable	Data type	Function
s	DINT, DWORD	area or equivalent constant for storing target value of high–speed counter
d	BOOL	available external output relay: Y0 to Y7

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	–	x	x	x	–	–	x
d	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
	–	x	–	–	–	–	–	–	–	–

x: available
–: not available

Example Below is an example of a ladder diagram (LD) body for the instruction.



F163_HC0R High–speed counter output reset

Steps 7

Description

Sets the value specified by **s** as target value of the high–speed counter if the trigger **EN** is in the ON–state. When the elapsed value (DT9045 and DT9044) of the high–speed counter agrees with the target value (DT9047 and DT9046), the external output relay specified by **d** turns OFF. You can use 8 external output relays (Y0 to Y7). When the F163 (HC0R) instruction is executed, the target value is stored in special data registers DT9047 and DT9046 and it is cleared when the elapsed value of the high–speed counter agrees with the target value. Use 24 bit binary data with sign data for the target value of HSC (FF800000 hex to 007FFFFF hex / –8,388,608 to 8,388,607). Once the F163 (HC0R) instruction is executed, special internal relay R903A turns ON and stays ON. It is cleared when the elapsed value of the high–speed counter agrees the target value. Even if the reset operation of the high–speed counter is performed after executing the F163 (HC0R) instruction, the target value setting is not cleared until the elapsed value of the high–speed counter agrees with the target value.

You can use the same external output relay specified by the F163 (HC0R) instruction in other parts of program. It is not considered duplicate use of the same output. While special internal relay R903A is in ON state, no other high–speed counter instructions F162_HC0S, F163 (HC0R), F164_SPD0, F165_CAM0 can be executed.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F163	–	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	DINT, DWORD	area or equivalent constant for storing target value of high–speed counter
d	BOOL	available external output relay: Y0 to Y7

Operands

For	Relay				T/C		Register			Constant
	DWX	DWY	DWR	DWL	DSV	DEV	DDT	DLD	DFL	dec. or hex.
s	x	x	x	–	x	x	x	–	–	x
d	**X**	**Y**	**R**	**L**	**T**	**C**	**DT**	**LD**	**FL**	**dec. or hex.**
	–	x	–	–	–	–	–	–	–	–

x: available
–: not available

Example

Below is an example of an instruction list (IL) body for the instruction.

```
LD           start           (*EN = start; Starting signal for the F163_HC0R function*)
F163_HC0R    Var_0, Var_1    (* s = Var_0*) (*  d = Var_1 *)
ST           out             (* option *)
```

F164_SPD0 Pulse output control; Pattern output control

Steps	3

Description Outputs the pattern of the pulse corresponding to the elapsed value of HSC. When the executing condition is ON and HSC control–flag (R903A) is OFF, this instruction starts operation. This instruction executes pattern output or pulse output corresponding to the data of the data table registered at the data register specified by **s**. You can use pulse output for positioning with a pulse motor and pattern output for controlling an inverter. When you execute pulse output with this instruction, input the pulse of Y7 directly to HSC or input the encoder output pulse. When you execute pattern output, input the encoder output pulse to HSC. Specify at system register No. 400 whether you will use HSC or not. It is not possible to execute this instruction without the following settings: input condition to detect a rising edge (0/1), and the HSC flag (R903A) must be reset.setting. The output coils of pattern output are within the 8 points Y0 to Y7. The output coil of pulse output is Y7 only. Select either pattern outputs or pulse outputs by the content of the first word of the data table. When you input 0 for one word of the first address (all bits are 0), it will be the pulse output. When you execute pattern output, an error occurs unless the No. of the control steps is 1 to F or unless the No. of control points is 1 to 8. An error occurs when the first target value is not FF800000 to 7FFFFF. An error does not occur when the first target value on and after the second one are not FF800000 to 7FFFFF. The operation, however, is stopped and R903A turns OFF. When the frequency data is ”0”, pulse output ends. It will also end if the area is exceeded during its execution.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F164	–	x	x	x	x

x: available
–: not available

Data types

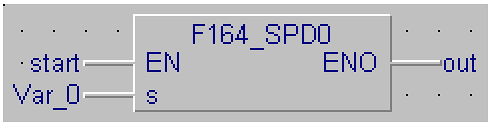
Variable	Data type	Function
s	INT, WORD	starting 16–bit area for storing control data

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	–	–	–	–	–	–	x	–	–	–

x: available
–: not available

Example Below is an example of a ladder diagram (LD) body for the instruction.



F165_CAM0 Cam control

Steps	3

Description Converts ON/OFF of output specified in the table corresponding to the elapsed value of HSC. This instruction controls the output up to 8 points (Y0 to Y7), corresponding to ON/OFF target value of each coil on the table, which is for the control of cam position specified by **s**. The target value is within the range of 23–bits data and 0 to 8388607 (i.e. 23 bits of data, 16#00000001 to 16#007FFFFF). If you execute cam control, you have to specify the operating mode as addition counter. (If it is not addition counter, you will not be able to execute the control properly.) The target value is maximum 32 steps with FP1–C16, maximum 64 steps with FP1–C24/C40. If you cancel hard reset, soft reset, and control maximum value you can set the initial pattern at output, set the elapsed value to 0 and restart Cam control. You can output the initial pattern at the start of control. However, you cannot clear the elapsed value to 0.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F165	–	x	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	starting 16–bit area for storing control data

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	–	–	–	–	–	–	x	–	–	–

x: available
–: not available

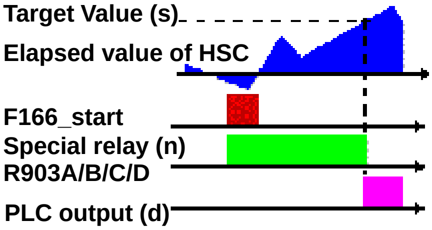
Example Below is an example of an instruction list (IL) body for the instruction.

```
LD          start       (*EN = start; Starting signal for the F165_CAM0 function*)
F165_CAM0   Var_0       (* s = Var_0*)
ST          out         (* option *)
```

F166_HC1S Sets Output of High–speed counter (4 Channels)

Steps 11

Description If the trigger EN of the instruction F166 has the status TRUE, pulses at the HSC will be counted. If the elapsed value of the high–speed counter equals the target value **s**, an interrupt will be executed and the output relay **d** of the PLC will be set. In addition to this the special relay for the HSC **n** (R903A/B/C/D) will be reset and F166 is deactivated.



If the high–speed counter is reset (reset input of HSC from 0 to 1, see system register 400/401 in the project navigator) before the elapsed value has reached the target value **s**, the elapsed value will be reset to zero. F166 remains active and counting restarts at zero.The duplicate use of an external output relay in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verifyed by FPWIN Pro and will not be detected. While the special relay(s) R903A/B/C/D is/are in ON state no other high–speed counter instructions can be executed.FP0 provides 4 HSC channels. The channel number is specified by n (0 to 3).

n values	**0**	**1**	**2**	**3**
Elapsed value register:	DDT9044	DDT9048	DDT9104	DDT9108
Target value register:	DDT9046	DDT9050	DDT9106	DDT9110
Used channel:	CH0 of HSC0	CH1 of HSC0	CH0 of HSC1	CH1 of HSC1
ON during execution:	R903A	R903B	R903C	R903D

s values	
FP0	FP–SIGMA
–8388608 or 16#FF800000	–2,147,483,648 or 16#80000000
...	...
8388607 or 16#7FFFFF	2,147,483,647 or 16#7FFFFFFF

d values	**output**
0	Y0
...	...
7	Y7

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F166	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
n	DINT, DWORD	the channel no. of the high–speed counter that corresponds to the matching output (n: 0 to 3)
s	DINT, DWORD	the high–speed counter target value data or the starting address of the area that contains the data
d	BOOL	the output coil that is turned on when the values match (Yn, n: 0 to 7)

Operands

For	Relay				T/C		Register			Constant
	DWX	**DWY**	**DWR**	**DWL**	**DSV**	**DEV**	**DDT**	**DLD**	**DFL**	**dec. or hex.**
n	–	–	–	–	–	–	–	–	–	x
s	x	x	x	–	x	x	x	–	–	x
d	**X**	**Y**	**R**	**L**	**T**	**C**	**DT**	**LD**	**FL**	**dec. or hex.**
	–	x	–	–	–	–	–	–	–	–

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	– index is too high – parameter s exceeds the valid range
R9008	%MX0.900.8	ON	

Example In this example the function F166_HC1S is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

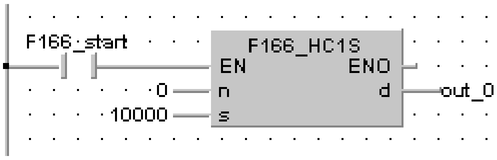
GVL In the **G**lobal **V**ariable **L**ist, you define variables that can be accessed by all POUs in the project.

	Identifier	Address	Type	Initial	Comment
0	out_0	%QX0.0	BOOL	FALSE	output Y0 of PLC

POU header In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_EXTERNAL	out_0	BOOL	FALSE	output Y0 of PLC
1	VAR	F166_start	BOOL	FALSE	F166 start condition

LD



IL

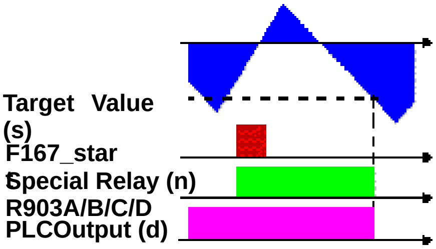
```
LD           F166_start          Load start condition
F166_HC1S    0,10000,out_0       execute F166
```

F167_HC1R Resets Output of High–speed Counter (4 Channels)

Steps 11

Description

If the trigger EN of the instruction F167 has the status TRUE, pulses at the HSC will be counted. If the elapsed value of the high–speed counter equals the target value **s**, an interrupt will be executed and the output relay **d** of the PLC will be reset. In addition to this the special relay for the HSC **n** (R903A/B/C/D) will be reset and F167 is deactivated.



If the high–speed counter is reset (reset input of HSC from 0 to 1, see system register 400/401 in the project navigator) before the elapsed value has reached the target value **s**, the elapsed value will be reset to zero. F167 remains active and counting restarts at zero. The duplicate use of an external output relay **d** in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verifyed by FPWIN Pro and will not be detected. While the special relay(s) R903A/B/C/D is/are in ON state no other high–speed counter instructions can be executed. FP0 provides 4 HSC channels. The channel number is specified by **n** (0 to 3).

n values	**0**	**1**	**2**	**3**
Elapsed value register:	DDT9044	DDT9048	DDT9104	DDT9108
Target value register:	DDT9046	DDT9050	DDT9106	DDT9110
Used channel:	CH0 of HSC0	CH1 of HSC0	CH0 of HSC1	CH1 of HSC1
ON during execution:	R903A	R903B	R903C	R903D

s values	
FP0	FP–SIGMA
–8388608 or 16#FF800000	–2,147,483,648 or 16#80000000
...	...
8388607 or 16#7FFFFF	2,147,483,647 or 16#7FFFFFFF

d values	**output**
0	Y0
...	...
7	Y7

PLC types

Availability	**FP0**	**FP1**		**FP–M**	
	2.7k, 5k, 10k	**0.9k**	**2.7k, 5k**	**0.9k**	**2.7k, 5k**
F167	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
n	DINT, DWORD	the channel no. of the high–speed counter that corresponds to the matching output (n: 0 to 3)
s	DINT, DWORD	the high–speed counter target value data or the starting address of the area that contains the data
d	BOOL	the output coil that is turned off when the values match (Yn, n: 0 to 7)

Operands

For	Relay				T/C		Register			Constant
	DWX	**DWY**	**DWR**	**DWL**	**DSV**	**DEV**	**DDT**	**DLD**	**DFL**	**dec. or hex.**
n	–	–	–	–	–	–	–	–	–	x
s	x	x	x	–	x	x	x	–	–	x
d	**X**	**Y**	**R**	**L**	**T**	**C**	**DT**	**LD**	**FL**	**dec. or hex.**
	–	x	–	–	–	–	–	–	–	–

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	ON	– index is too high –parameter s exceeds the valid range
R9008	%MX0.900.8	ON	

Example

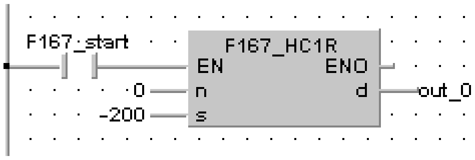
In this example the function F167_CMPR is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

	Class	Identifier	Type	Initial	Comment
0	VAR_	out_0	BOOL	FALSE	output Y0 of PLC
1	VAR	F167_start	BOOL	FALSE	F167 start condition

LD



IL

```
LD          F167_start       load start condition
F167_HC1R   0,–200,out_0     execute F167
```

F168_SPD1

Positioning pulse instruction

Steps	5

Description

When the corresponding control flag is off and the execution condition (trigger) is in the on state, a pulse is output from the specified output (Y0 or Y1).

The control code, initial speed, maximum speed, acceleration/deceleration time, and target value, are specified by using a Data Unit Type (DUT).

The frequency is switched by the acceleration/deceleration time specified for changing from the initial speed to the maximum speed.

See below for the corresponding areas:

Channel no.	Control flag	Elapsed value area	Target value area	Direction-al output	Near home	Home input
ch0	R903A	DDT9044	DDT9046	Y2	DT9052 bit2	X0
ch1	R903B	DDT9048	DDT9050	Y3	DT9052 bit6	X1

☞

- **When this instruction is used, the setting for the channel corresponding to system register 400 should be set to “High–speed counter not used”.**
- **By performing rewrite during RUN during pulse output, more than the set number of pulses may be output.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F168	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	Data Unit Type (DUT)	starting address for the area that contains the data table
n*	decimal constant	output Yn that corresponds to the pulse output (n: 0 or 1)

Operands

For	Relay			T/C		Register			Constant
	WX	WY	WR	SV	EV	DT	LD	FL	dec. or hex.
s	–	–	–	–	–	x	–	–	–
n*	–	–	–	–	–	–	–	–	x

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– n* is a number other than 0 or 1 – the control code exceeds the limit of specified range – Initial Speed Fmin < 40 – Initial Speed Fmin > Maximum Speed Fmax – Target Value (pulse number) exceeds the limit of specified range
R9008	%MX0.900.8	for an instant	

Description of operating mode

Incremental <relative value control>

Outputs the pulse set by the target value.

By setting 16#02 (incremental; forward: off; reverse: on) in the control code, when the target value is positive, the directional output is turned off and the elapsed value of the high–speed counter increases. When the target value is negative, the directional output turns on and the elapsed value of the high–speed counter decreases. By setting 16#03 in the control code, the directional output is the reverse of that above.

Absolute <absolute value control>

Outputs the pulse set by the difference between the current value and the target value. (The difference between the current value and the target value is the output pulse number.)

By setting 16#12 (absolute; forward: off; reverse: on) in the control code, when the current value is less than the target value, the directional output is turned off and the elapsed value of the high–speed counter increases. When the current value is greater than the target value, the directional output turns on and the elapsed value of the high–speed counter decreases. By setting 16#13 in the control code, the directional output is the reverse of that above.

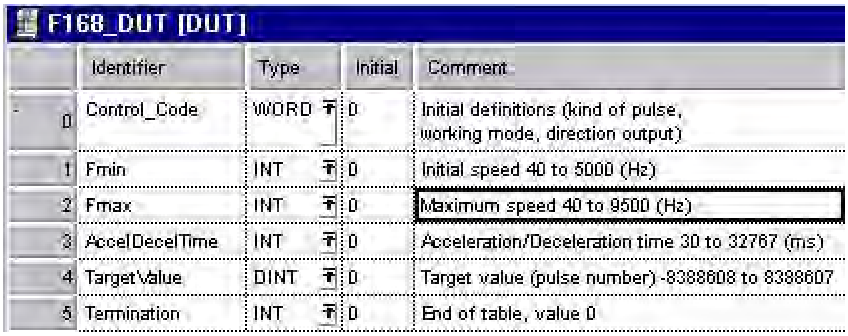
Home return

Until the home input (X0 or X1) is entered, the pulse is continuously output. To decelerate the movement when near the home, set the bit corresponding to DT9052 to off → on → off → with the near home input.

To return to the home, refer to only the control code, initial speed, maximum speed, and acceleration/deceleration time of the data table.

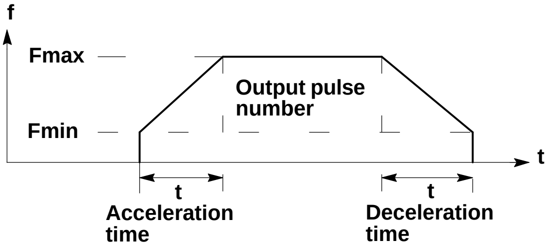
During operation, the elapsed value area and set value area will become insufficient. At the completion of operations, the elapsed value will become 0.

Data Unit Type settings



F168_DUT [DUT]

	Identifier	Type	Initial	Comment
0	Control_Code	WORD	0	Initial definitions (kind of pulse, working mode, direction output)
1	Fmin	INT	0	Initial speed 40 to 5000 (Hz)
2	Fmax	INT	0	Maximum speed 40 to 9500 (Hz)
3	AccelDecelTime	INT	0	Acceleration/Deceleration time 30 to 32767 (ms)
4	TargetValue	DINT	0	Target value (pulse number) -8388608 to 8388607
5	Termination	INT	0	End of table, value 0



1) Specify the control code (line 0 in DUT above).

16# □ □ □

Pulse width specification

0: Duty 50%

1: Fixed pulse width (approx. 80μs)

Operation mode and directional output theory

00: Does not use incremental directional output
02: Incremental forward off/reverse on
03: Incremental forward on/reverse off
10: Does not use absolute directional output
12: Absolute forward off/reverse on
13: Absolute forward on/reverse off
20: No home return directional output
22: Home return directional output off
23: Home return directional output on
24: No home return directional output
(Home input valid only after near home input.)
26: Home return output off
(Home input valid only after near home input.)
27: Home return output on
(Home input valid only after near home input.)

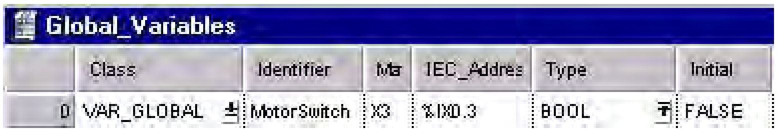
(☛ 24, 26, and 27 are supported by CPU Ver. 2.0 and subsequent versions.)

2) When the pulse width is set to duty 50%, the maximum is 6kHz. When the pulse width is set to fixed pulse width (approx. 80μs), the maximum is 9.5kHz (line 2 in DUT above).

3) The Target Value and Termination specifications are not necessary when a home return is carried out (lines 4 and 5 in DUT above).

Example In this example the function F168_SPD1 is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

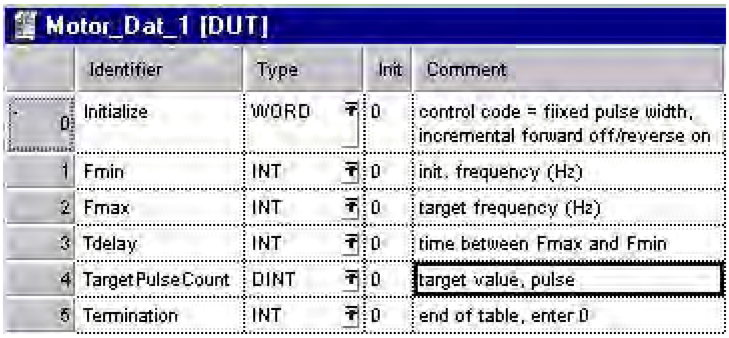
GVL In the **G**lobal **V**ariable **L**ist, you define variables that can be accessed by all POUs in the project.



Global_Variables

	Class	Identifier	Ma	IEC_Addres	Type	Initial
0	VAR_GLOBAL	MotorSwitch	X3	%IX0.3	BOOL	FALSE

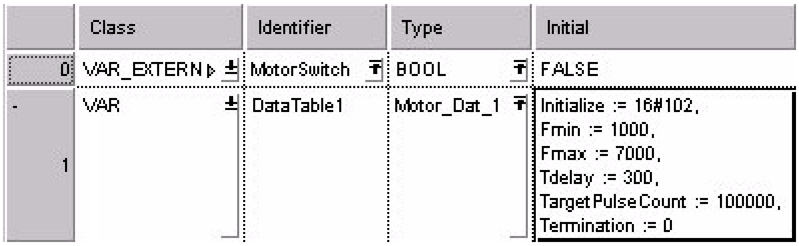
DUT With a **D**ata **U**nit **T**ype you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT_Pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.



Motor_Dat_1 [DUT]

	Identifier	Type	Init	Comment
0	Initialize	WORD	0	control code = fiixed pulse width, incremental forward off/reverse on
1	Fmin	INT	0	init. frequency (Hz)
2	Fmax	INT	0	target frequency (Hz)
3	Tdelay	INT	0	time between Fmax and Fmin
4	TargetPulseCount	DINT	0	target value, pulse
5	Termination	INT	0	end of table, enter 0

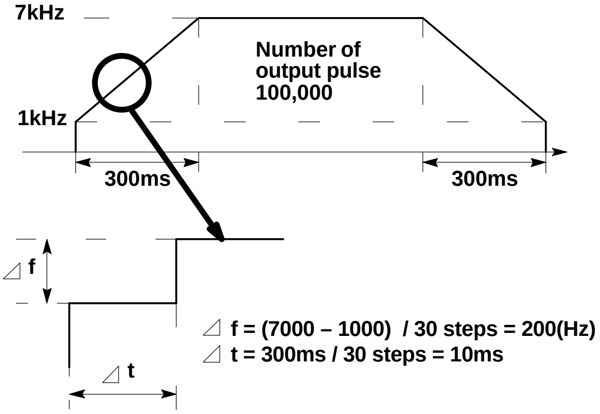
POU header In the POU header, all input and output variables are declared that are used for programming this function.



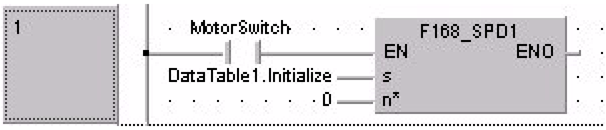
	Class	Identifier	Type	Initial
0	VAR_EXTERN	MotorSwitch	BOOL	FALSE

1	VAR	DataTable1	Motor_Dat_1	Initialize := 16#102,
				Fmin := 1000,
				Fmax := 7000,
				Tdelay := 300,
				TargetPulseCount := 100000,
				Termination := 0

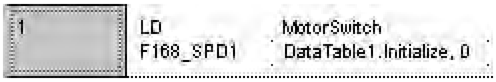
Body The parameters defined in the DUT will be executed in the body as illustrated below:


7kHz
Number of output pulse 100,000
1kHz
300ms
300ms
⊿ f
⊿ t
⊿ f = (7000 – 1000) / 30 steps = 200(Hz)
⊿ t = 300ms / 30 steps = 10ms

LD

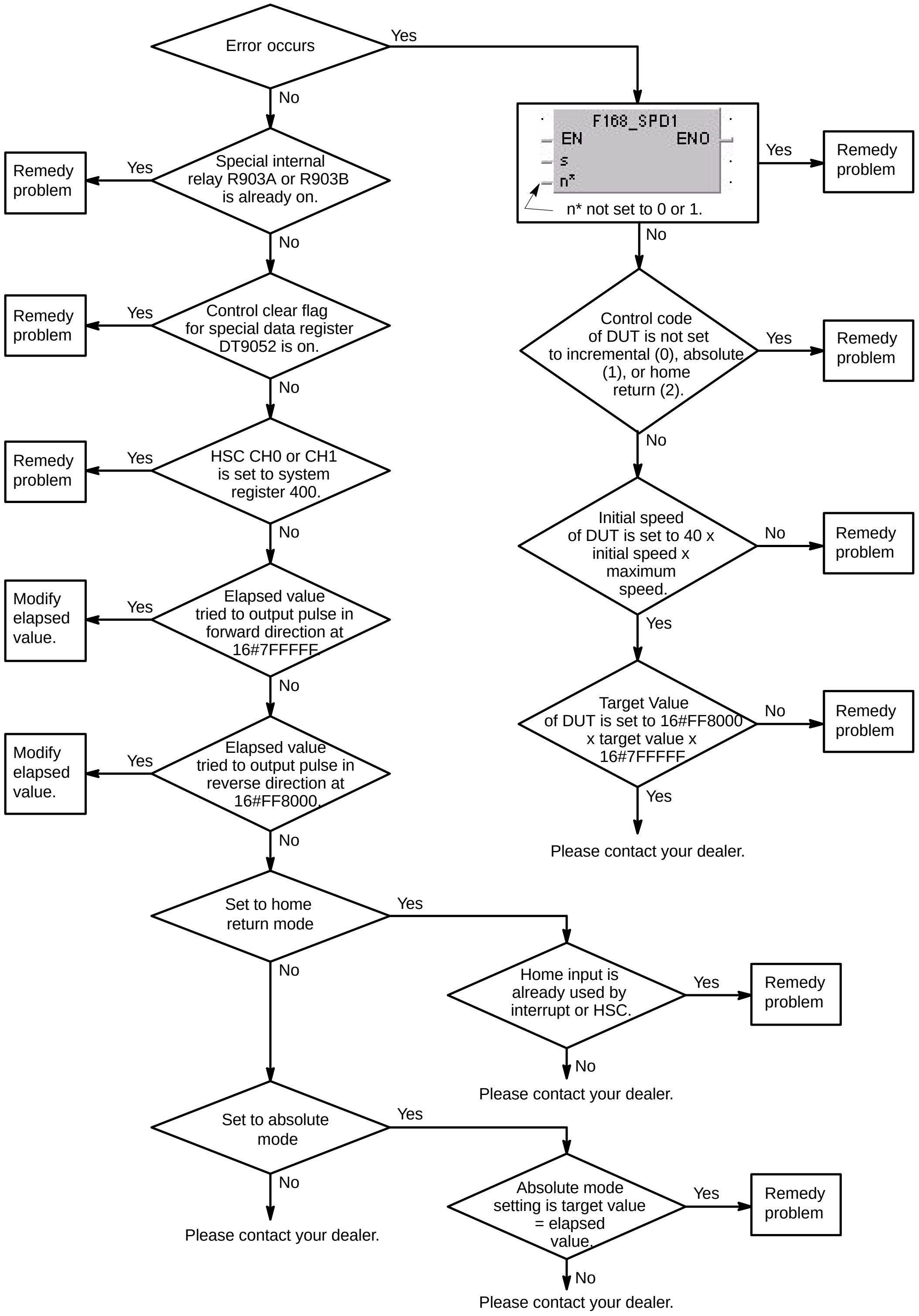


IL



```
1    LD          MotorSwitch
     F168_SPD1   DataTable1.Initialize, 0
```

Troubleshooting flowchart if a pulse is not output when instruction F168_SPD1 is executed


Error occurs
Yes
No
Special internal relay R903A or R903B is already on.
Yes
Remedy problem
No
Control clear flag for special data register DT9052 is on.
Yes
Remedy problem
No
HSC CH0 or CH1 is set to system register 400.
Yes
Remedy problem
No
Elapsed value tried to output pulse in forward direction at 16#7FFFFF.
Yes
Modify elapsed value.
No
Elapsed value tried to output pulse in reverse direction at 16#FF8000.
Yes
Modify elapsed value.
No
Set to home return mode
Yes
Home input is already used by interrupt or HSC.
Yes
Remedy problem
No
Please contact your dealer.
No
Set to absolute mode
Yes
Absolute mode setting is target value = elapsed value.
Yes
Remedy problem
No
Please contact your dealer.
No
Please contact your dealer.
F168_SPD1
EN
ENO
s
n*
n* not set to 0 or 1.
Yes
Remedy problem
No
Control code of DUT is not set to incremental (0), absolute (1), or home return (2).
Yes
Remedy problem
No
Initial speed of DUT is set to 40 x initial speed x maximum speed.
No
Remedy problem
Yes
Target Value of DUT is set to 16#FF8000 x target value x 16#7FFFFF.
No
Remedy problem
Yes
Please contact your dealer.

F169_PLS

Pulse width modulation >= 40 Hz

Steps 5

Description

When the corresponding control flag is off and the execution condition (trigger) is in the on state, a pulse is output from the specified channel. The pulse is output while the execution condition (trigger) is in the on state.

By specifying either incremental counting or decremental counting in the control code, this instruction can be used as an instruction for JOG operations. For that situation, set the control code with combinations such as 16#xx12 (incremental, directional output off) and 16#xx22 (decremental, directional output on).

The frequency and duty can be changed each scan. (This becomes effective with the next pulse output after this instruction is executed.)

See below for the corresponding areas.

Channel no.	Control flag	Data register for elapsed value
ch0	R903A	DDT9044
ch1	R903B	DDT9048

When using the incremental counting mode, the pulse stops when the elapsed value exceeds 16#7FFFFF.

When using the decremental counting mode, the pulse stops when the elapsed value exceeds 16#FF800000.

☞

- **When this instruction is used, the setting for the channel corresponding to system register 400 should be set to “High–speed counter not used”.**
- **By performing a rewrite during RUN while operating, the pulse output will stop during rewriting.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F169	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	ARRAY[0..1] of INT or WORD	data table
n*	decimal constant	output Yn that corresponds to the pulse output (n: 0 or 1)

Operands

For	Relay			T/C		Register			Constant
	WX	WY	WR	SV	EV	DT	LD	FL	dec. or hex.
s	–	–	–	–	–	x	–	–	–
n*	–	–	–	–	–	–	–	–	x

x: available
–: not available

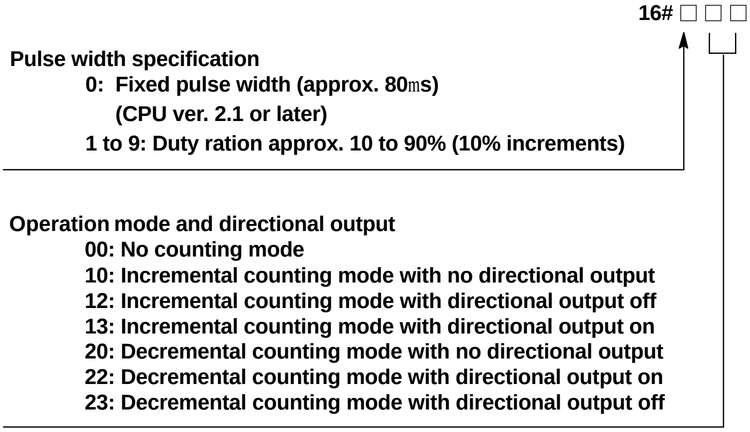
Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– n* is a number other than 0 or 1
R9008	%MX0.900.8	for an instant	

■ Data table settings

ARRAY[0]	Control code	(*1)
ARRAY[1]	Frequency (Hz)	40 to 10000 (Hz) (*2)

1) Specify the control code.



16# □ □ □

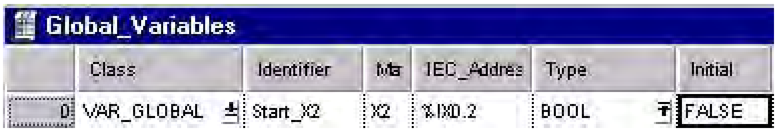
Pulse width specification
0: Fixed pulse width (approx. 80ms)
(CPU ver. 2.1 or later)
1 to 9: Duty ration approx. 10 to 90% (10% increments)

Operation mode and directional output
00: No counting mode
10: Incremental counting mode with no directional output
12: Incremental counting mode with directional output off
13: Incremental counting mode with directional output on
20: Decremental counting mode with no directional output
22: Decremental counting mode with directional output on
23: Decremental counting mode with directional output off

2) Frequency setting range: 40 to 10000 (Hz)
If “0 to 39” is set in ARRAY[1], the frequency is set to 40Hz (40).

Example In this example the function F169_PLS is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

GVL In the **G**lobal **V**ariable **L**ist, you define variables that can be accessed by all POUs in the project.

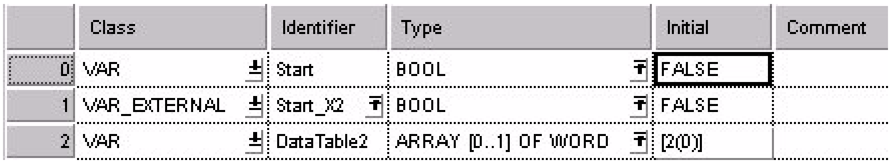


Global_Variables

	Class	Identifier	FP	IEC_Address	Type	Initial
0	VAR_GLOBAL	Start_X2	X2	%IX0.2	BOOL	FALSE

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

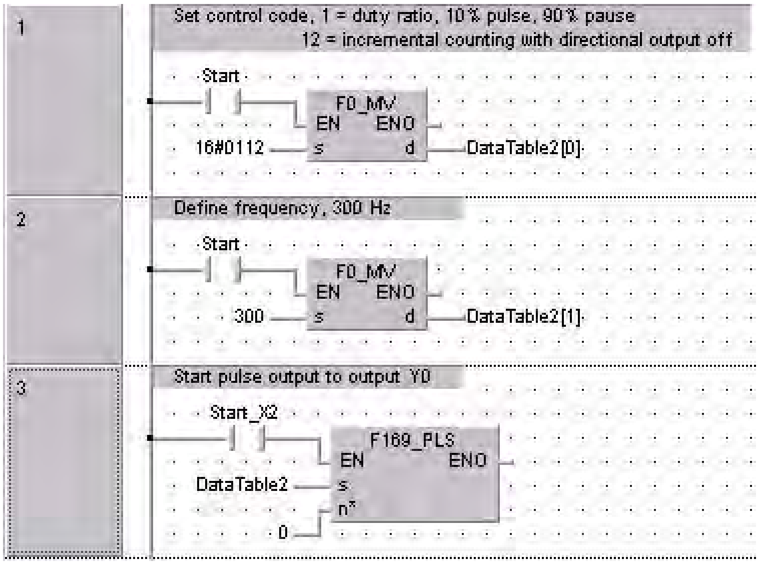


	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR_EXTERNAL	Start_X2	BOOL	FALSE	
2	VAR	DataTable2	ARRAY [0..1] OF WORD	[2(0)]	

Body

The comment fields in the LD and IL bodies explain the function of this example.

LD

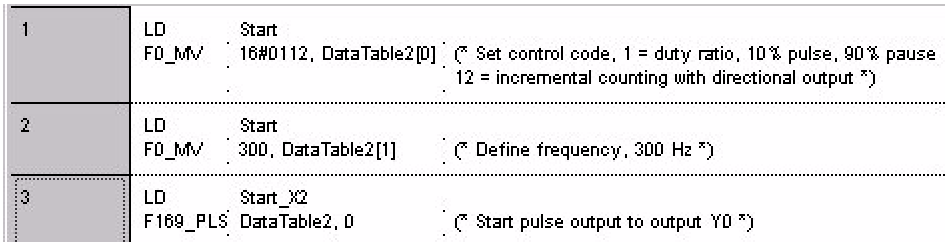


1 Set control code, 1 = duty ratio, 10% pulse, 90% pause
12 = incremental counting with directional output off

2 Define frequency, 300 Hz

3 Start pulse output to output Y0

IL



```
1   LD        Start
    F0_MV     16#0112, DataTable2[0]  (* Set control code, 1 = duty ratio, 10% pulse, 90% pause
                                         12 = incremental counting with directional output *)

2   LD        Start
    F0_MV     300, DataTable2[1]      (* Define frequency, 300 Hz *)

3   LD        Start_X2
    F169_PLS  DataTable2, 0           (* Start pulse output to output Y0 *)
```

F170_PWM

Pulse width modulation

Steps	5

Description

When the corresponding control flag is off and execution condition (trigger) is in the on state, a PWM is output from the specified channel. The PWM is output while the execution condition (trigger) is in the on state.

The frequency and duty are specified with the data table.

Since the output is delayed near the maximum and minimum levels, the set duty ratio will differ.

The duty can be changed each scan. The frequency settings is only effective at the start of the execution of the instruction (becomes effective after the next pulse output).

See below for the corresponding areas.

Channel no.	Control flag
ch0	R903A
ch1	R903B

☞

- **When this instruction is used, the setting for the channel corresponding to system register 400 should be set to “High–speed counter not used”.**
- **By performing a rewrite during RUN while operating, the pulse output will stop during rewriting.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F170	x	–	–	–	–

x: available
–: not available

Data types

Variable	Data type	Function
s	ARRAY[0..1] of INT or WORD	data table
n*	decimal constant	output Yn that corresponds to the pulse output (n: 0 or 1)

Operands

For	Relay			T/C		Register			Constant
	WX	WY	WR	SV	EV	DT	LD	FL	dec. or hex.
s	–	–	–	–	–	x	–	–	–
n*	–	–	–	–	–	–	–	–	x

x: available
–: not available

Error flags

No.	IEC address	Set	If
R9007	%MX0.900.7	permanently	– n* is a number other than 0 or 1 – the frequency setting value set with the control code (ARRAY[0]) is outside the specification range – 100% or higher is set with Duty (ARRAY[1])
R9008	%MX0.900.8	for an instant	

Data table settings

ARRAY[0]	**Control code**	**16#0 to 16#8, 16#11 to 16#16 (*1)**
ARRAY[1]	**Duty (%)**	**1 to 999 (0.1% to 99.9%)**

1) Control code contents (frequency settings)

16#11: Frequency 1 kHz	(Cycle 1.0ms)
16#12: Frequency 714 Hz	(Cycle 1.25ms)
16#13: Frequency 500 Hz	(Cycle 2.0ms)
16#14: Frequency 400Hz	(Cycle 2.5ms)
16#15: Frequency 200 Hz	(Cycle 5.0ms)
16#16: Frequency 100 Hz	(Cycle 10ms)
16#0: Frequency 38 Hz	(Cycle 26ms)
16#1: Frequency 19 Hz	(Cycle 52ms)
16#2: Frequency 9.5 Hz	(Cycle 105ms)
16#3: Frequency 4.8 Hz	(Cycle 210ms)
16#4: Frequency 2.4 Hz	(Cycle 420ms)
16#5: Frequency 1.2 Hz	(Cycle 840ms)
16#6: Frequency 0.6 Hz	(Cycle 1.6s)
16#7: Frequency 0.3 Hz	(Cycle 3.4s)
16#8: Frequency 0.15 Hz	(Cycle 6.7s)

☛ 16#11 to 16#16 are supported by CPU Ver. 2.0 and subsequent versions.

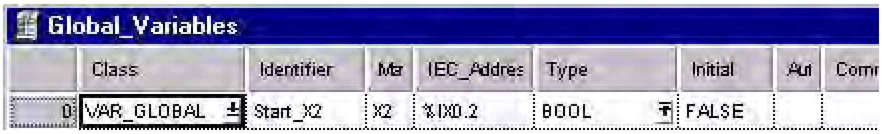
ARRAY[1] –> pulse width

the table below shows all possible values for the first ARRAY element:

ARRAY[1]	ON TIME	OFF TIME	
0	0 %ON	100 %OFF	The pulse width (ON/OFF time) ca be changed during execution of F170. The changes are valid after the current period is finished .
1	0.1 %ON	99.9 %OFF	
2	0.2 %ON	99.8 %OFF	
...	...	... %OFF	
998	99.8 %ON	0.2 %OFF	
999	99.9 %ON	0.1 %OFF	
1000	100 %ON	0 %OFF	

Example In this example the function F170_PWM is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

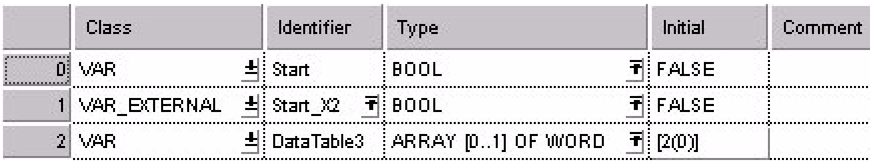
GVL In the **G**lobal **V**ariable **L**ist, you define variables that can be accessed by all POUs in the project.



Global_Variables

	Class	Identifier	Ma	IEC_Addres	Type	Initial	Aut	Comr
0	VAR_GLOBAL	Start_X2	X2	%IX0.2	BOOL	FALSE		

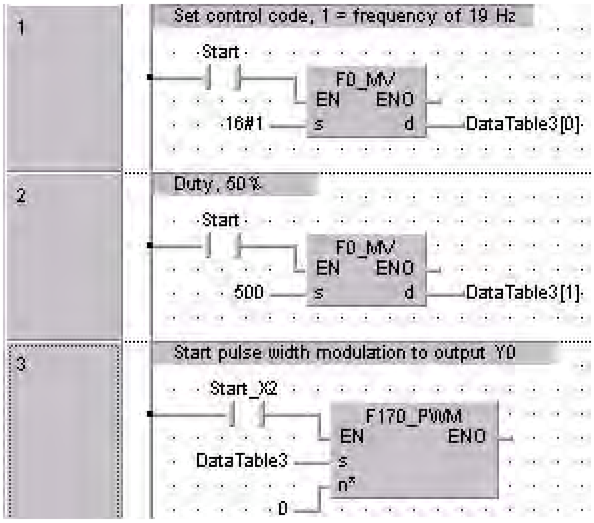
POU header In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	Start	BOOL	FALSE	
1	VAR_EXTERNAL	Start_X2	BOOL	FALSE	
2	VAR	DataTable3	ARRAY [0..1] OF WORD	[2(0)]	

Body The comment fields in the LD and IL bodies explain the function of this example.

LD

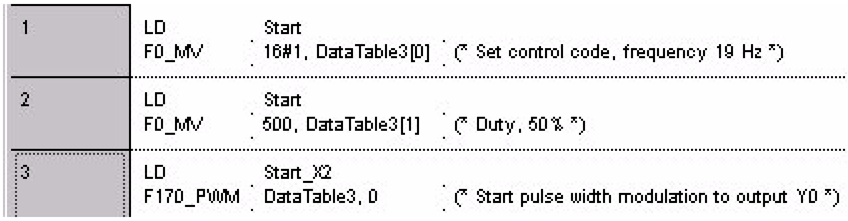


1 Set control code, 1 = frequency of 19 Hz

2 Duty, 50 %

3 Start pulse width modulation to output Y0

IL



```
1   LD        Start
    F0_MV     16#1, DataTable3[0]   (* Set control code, frequency 19 Hz *)
2   LD        Start
    F0_MV     500, DataTable3[1]    (* Duty, 50 % *)
3   LD        Start_X2
    F170_PWM  DataTable3, 0         (* Start pulse width modulation to output Y0 *)
```

Chapter 26

Basic Sequence Instructions

DF

Leading edge differential

Steps	1

Description **DF** is a leading edge differential instruction. The **DF** instruction executes and turns ON output **o** for a singular scan duration if the trigger **i** changes from an OFF to an ON state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
DF	x	x	x	x	x

x: available
–: not available

Data types

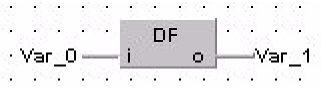
Variable	Data type
BOOL	input
BOOL	output

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
i	x	x	x	x	x	x	–	–	–	–
o	–	x	x	x	–	–	–	–	–	–

x: available
–: not available

Example Below is an example of a ladder diagram (LD) body for the instruction.



DFN

Trailing edge differential

Steps	1

Description

DFN is trailing edge differential instruction. The **DFN** instruction executes and turns ON output **o** for a singular scan duration if the trigger **i** changes from an ON to an OFF state.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
DFN	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type
BOOL	input
BOOL	output

Operands

For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
i	x	x	x	x	x	x	–	–	–	–
o	–	x	x	x	–	–	–	–	–	–

x: available
–: not available

Example

Below is an example of an instruction list (IL) body for the instruction.

```
LD       Var_0    (* i = Var_0 *)
DFN               (* Trailing edge differential for
                  variable Var_0. *)
ST       Var_1    (* o = Var_1 *)
                  (* At valid event the output
                  variable Var_1 *)
                  (* is in the ON–state for one scan
                  duration. *)
```

KEEP

Keep output ON or OFF depending on input variables

Steps	1

Description **KEEP** serves as a relay with set and reset points. When the **SetTrigger** turns ON, output of the specified relay goes ON and maintains its condition. Output relay goes OFF when the **ResetTrigger** turns ON. The output relay’s ON state is maintained until a **ResetTrigger** turns ON regardless of the ON or OFF states of the **SetTrigger**. If the **SetTrigger** and **ResetTrigger** turn ON simultaneously, the **ResetTrigger** is given priority.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
KEEP	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
Set Trigger	BOOL	sets Address output, i.e. turns in ON
Reset Trigger	BOOL	resets Address output, i.e. turns it OFF
Address	BOOL	specifed relay whose status (set or reset) is kept

Operands

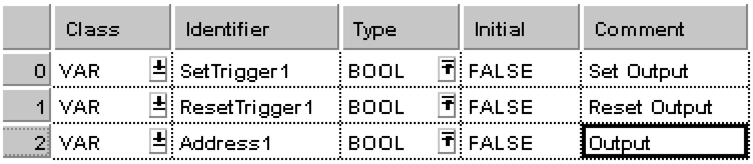
For	Relay				T/C		Register			Constant
	X	Y	R	L	T	C	DT	LD	FL	dec. or hex.
SetTrigger **ResetTrigger**	x	x	x	x	x	x	–	–	–	–
Address	–	x	x	x	–	–	–	–	–	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

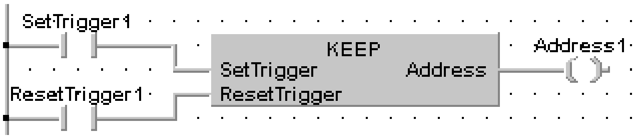
POU header

In the POU header, all input and output variables are declared that are used for programming this function.



	Class	Identifier	Type	Initial	Comment
0	VAR	SetTrigger1	BOOL	FALSE	Set Output
1	VAR	ResetTrigger1	BOOL	FALSE	Reset Output
2	VAR	Address1	BOOL	FALSE	Output

LD



ST

```
Address1:=KEEP(SetTrigger1, ResetTrigger1);
```

SET, RST

Set, Reset

Steps	3

Description

SET: When the execution conditions have been satisfied, the output is turned on, and the on status is retained.
RST: When the execution conditions have been satisfied, the output is turned off, and the off status is retained.

- You can use relays with the same number as many times as you like with the **SET** and **RST** instructions. (Even if a total check is run, this is not handled as a syntax error.)
- When the **SET** and **RST** instructions are used, the output changes with each step during processing of the operation.
- To output a result while operation is still in progress, use a partial I/O update instruction (**F143**).
- The output destination of a **SET** instruction is held even during the operation of an **MC** instruction.
- The output destination of a **SET** instruction is reset when the mode is changed from RUN to PROG. or when the power is turned off, except when a hold type internal relay is specified as the output destination.
- Placing a **DF** instruction (or specifying a rising edge in LD) before the **SET** and **RST** instructions ensures that the instruction is only executed at a rising edge.
- Relays:
  - Relays can be turned off using the **RST** instruction.
  - Using the various relays with the **SET** and **RST** instructions does not result in double output.
  - It is not possible to specify a pulse relay (P) as the output destination for a **SET** or **RST** instruction.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
SET, RST	x	x	x	x	x

x: available
–: not available

Operands

Instruction	Relay						T/C		Constant
	X	Y	R	L	E	P	SV	EV	dec. or hex.
SET, RST	–	x	x	x	x	–	–	–	–

x: available
–: not available

Example In this example, the SET and RESET instructions are demonstrated in function block diagram (FBD), ladder diagram (LD) and instruction list (IL). Since addresses are assigned directly using Matsushita addresses, no POU header is necessary.

Body Using the DF command or specifying a rising edge refines the program by making the programming step valid for one scan only:

1) When the input X0 is activated, the output Y0 is set.

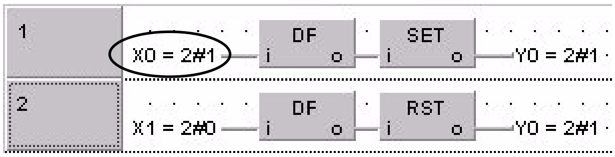
2) When the input X0 is turned off, the output Y0 remains set.

3) When the input X1 is activated, the output Y0 is reset.

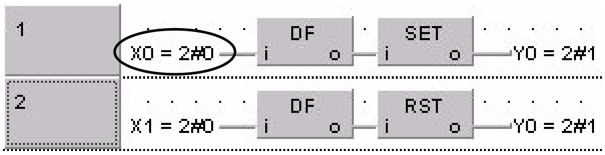
4) When the input X0 is reactivated, the output Y0 is set.

FBD

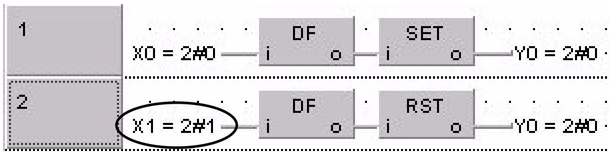
1)



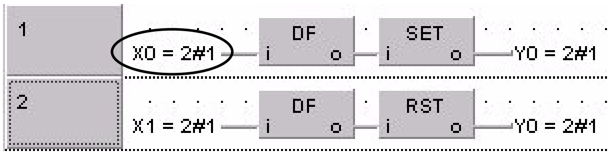
2)



3)

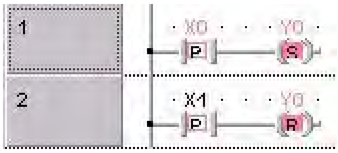


4)

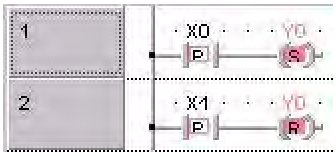


LD In ladder diagram, specify a rising edge in the contact and SET or RESET in the coil:

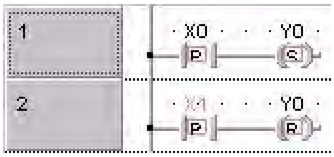
1)



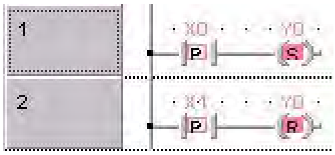
2)



3)

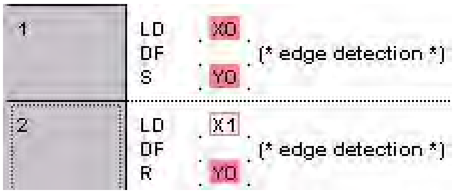


4)



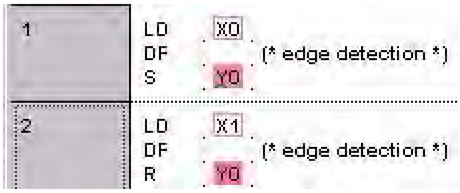
IL In instruction list, S and R are used for SET and RESET:

1)



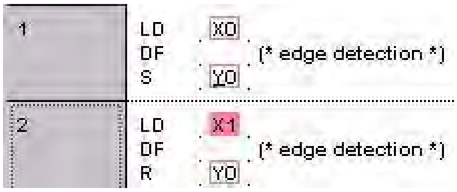
```
1   LD   X0
    DF        (* edge detection *)
    S    Y0
2   LD   X1
    DF        (* edge detection *)
    R    Y0
```

2)



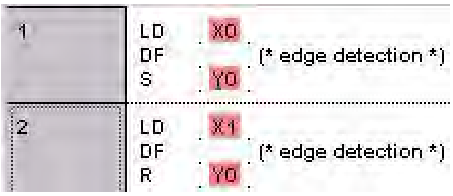
```
1   LD   X0
    DF        (* edge detection *)
    S    Y0
2   LD   X1
    DF        (* edge detection *)
    R    Y0
```

3)



```
1   LD   X0
    DF        (* edge detection *)
    S    Y0
2   LD   X1
    DF        (* edge detection *)
    R    Y0
```

4)



```
1   LD   X0
    DF        (* edge detection *)
    S    Y0
2   LD   X1
    DF        (* edge detection *)
    R    Y0
```

Chapter 27

Control Instructions

MC

Master control relay

Steps 2

Description

The **MC** (Master Control Relay) instruction executes the program between the master control relay **MC** and master control relay end **MCE** instructions of the same number **Num*** only if the trigger **EN** is in the ON–state. When the predetermined trigger **EN** is in the OFF state, the program between the master control relay **MC** and master control relay end **MCE** instructions are not executed. A master control instruction **(MC** and **MCE)** pair may also be programmed in between another pair of master control instructions. This construction is called ”nesting”.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
MC	x	x	x	x	x

x: available
–: not available

Data types

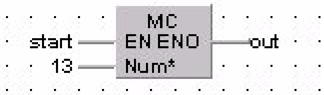
Variable	Data type	Function
Num*	constant	Constant number that must correspond to MCE number, both of which delimit a “nested” program that is not executed

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

Example

Below is an example of a ladder diagram (LD) body for the instruction.



MCE

Master control relay end

Steps	2

Description The **MCE** (Master Control Relay End) instruction executes the program between the master control relay **MC** and master control relay end **MCE** instructions of the same number **Num*** only if the trigger **EN** is in the ON–state. When the predetermined trigger **EN** is in the OFF state, the program between the master control relay **MC** and master control relay end **MCE** instructions are not executed. A master control instruction **(MC** and **MCE)** pair may also be programmed in between another pair of master control instructions. This construction is called ”nesting”.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
MCE	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to MC number, both of which delimit a “nested” program that is not executed

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

Example Below is an example of an instruction list (IL) body for the instruction.

```
LD     start  (* EN = start; Starting signal for
              the MC/MCE function. *)
MC     1      (* 1 = Num* *)
              (* ... *)
              (* Execute or execute not this
              program part. *)
              (* ... *)
MCE    1      (* 1 = Num* *)
```

JP

Jump

Steps 2

Description

The **JP** (Jump to Label) instruction skips to the Label (LBL) function that has the same number **Num*** as the **JP** function when the predetermined trigger **EN** is in the ON–state. The **JP** function will skip all instructions between a **JP** and an **LBL** of the same number. When the **JP** instruction is executed, the execution time of the skipped instructions is not included in the scan time. Two or more **JP** functions with the same number **Num*** can be used in a program. However, no two **LBL** instructions may be identically numbered. LBL instructions are specified as destinations of **JP**, **LOOP** and **F19_SJP** instructions. One **JP** and **LBL** instruction pair can be programmed between another pair. This construction is called nesting.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
JP	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to LBL number, this “nested” program is jumped over

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

Example

Below is an example of an instruction list (IL) body for the instruction.

```
LD     start   (* EN = start; Starting signal for
               the JP function. *)
JP     1       (* Num* = 1 (Address of Label) *)
```

LOOP

Loop

Steps 4

Description

The **LOOP** (Loop to Label) instruction skips to the **LBL** instruction with the same number **Num*** as the **LOOP** instruction and repeats execution of what follows until the data of a specified operand becomes ”0”. The **LBL** instructions are specified as destination of the **LOOP** instruction. It is not possible to specify two or more **LBL** instructions with the same number **Num*** within a program. If the set value **s** in the data area is ”0” from the beginning, the **LOOP** instruction is not executed (ignored).

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
LOOP	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	INT, WORD	Set value
Num*	constant	Constant number that must correspond to LBL number, this “nested” program is looped until the variable at s reaches 0

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s	x	x	x	x	x	x	x	x	x	–

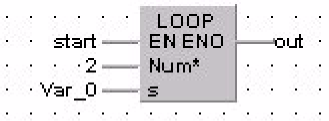
x: available
–: not available

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

Example

Below is an example of a ladder diagram (LD) body for the instruction.



LBL

Label

Steps	1

Description The **LBL** (Label for the JP and LOOP) instruction skips to the **LBL** instruction with the same number **Num*** as the JUMP instruction if the predetermined trigger **EN** is in the ON–state. It skips to the **LBL** instruction with the same number **Num*** as the LOOP instruction and repeats execution of what follows until the data of a specified operand becomes ”0”.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
LBL	x	x	x	x	x

x: available
–: not available

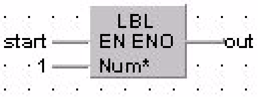
Data types

Variable	Data type	Function
Num*	constant	Constant number that must correspond to JP, LOOP or F19 label number

☞

- **It is not possible to use this function in a function block POU.**
- **The maximum possible value that can be assigned to Num* depends on the PLC type.**

Example Below is an example of a ladder diagram (LD) body for the instruction.



ICTL

Interrupt control

Steps	5

Description

The **ICTL** (Interrupt Control) instruction sets all interrupts to enable or disable. Each time the ICTL instruction is executed, it is possible to set parameters like the type and validity of interrupt programs. Settings can be specified by **s1** and **s2**.

- s1: 16–bit equivalent constant or 16–bit area for interrupt control setting
- s2: 16–bit equivalent constant or 16–bit area for interrupt trigger condition setting

The number of interrupt programs available is:

- 16 interrupt module initiated interrupt programs (INT 0 to INT 15)
- 8 advanced module (special modules, like positioning,...) initiated interrupt programs (INT 16 to INT 23)
- 1 time–initiated interrupt program (INT 24) (Time base 0.5 ms and 10ms selectable for FP10SH)

Be sure to use ICTL instructions so that they are executed once at the leading edge of the ICTL trigger using the DF instruction. Two or more ICTL instructions can have the same trigger.

Bit	15 .. 8	7 .. 0
s1 16#	**Selection of control function** **00**: Interrupt ”enable/disable” control **01**: Interrupt trigger reset control	**Interrupt type selection** **00**: Interrupt module initiated interrupt (INT 0–15) **01**: Advanced module initiated interrupt (INT 16–23) **02**: Time–initiated interrupt (INT 24)
s1 16# **s2 2#**	**00** Bit 0: **0** Interrupt program 0 disabled Bit 0: **1** Interrupt program 0 enabled Bit 1: **0** Interrupt program 1 disabled ... Bit 15: **1** Interrupt program 15 enabled ✌ Example: **s2** = **2#0000000000001010**	**00**

☞

- **The current enable/disable status of each interrupt module initiated interrupt can be checked by monitoring the special data register DT90025.**
- **The current enable/disable status of each non–interrupt module initiated interrupt can be checked by monitoring the special data register DT90026.**
- **The current interrupt interval of the time–interrupt can be checked by monitoring the special data register DT90027.**
- **If a program is written into an interrupt task, the interrupt concerned will be enabled automatically during the initialization routine when starting the program.**

- **With the ICTL instruction an interrupt task can be enabled or disabled by the program.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
ICTL	–	–	x	–	x

x: available
–: not available

Data types

Variable	Data type	Function
s1	INT, WORD	Interrupt control data setting
s2	INT, WORD	Interrupt condition setting

Operands

For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
s1, s2	–	x	x	x	x	x	x	x	x	x

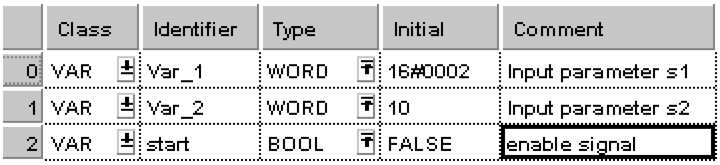
x: available
–: not available

Example

In this example the function ICTL is programmed in ladder diagram (LD) and instruction list (IL). The same POU header is used for both programming languages.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

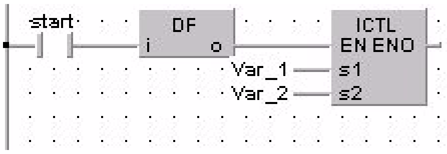


	Class	Identifier	Type	Initial	Comment
0	VAR	Var_1	WORD	16#0002	Input parameter s1
1	VAR	Var_2	WORD	10	Input parameter s2
2	VAR	start	BOOL	FALSE	enable signal

Body

The interval for executing INT 24 program is specified as 100 ms (10ms time base selected) when the leading edge of start is detected.

LD



IL

```
LD    start        (* Load value of EN–input*)
DF                 (* Leading edge detection *)
ICTL  Var_1,Var_2  (* Execute ICTL *)
```

Chapter 28

Special Instructions

F140_STC Carry–flag set

Steps	1

Description Special internal relay R9009 (carry–flag) goes ON if the trigger **EN** is in the ON–state. This instruction can be used to control data using carry–flag R9009, e.g. F122_RCR and F123_RCL instructions.

PLC types

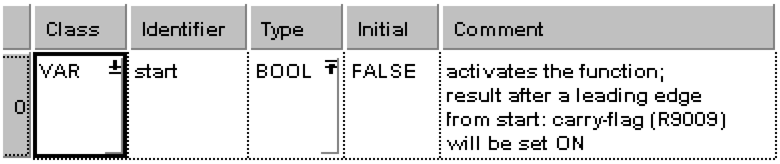
Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F140	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

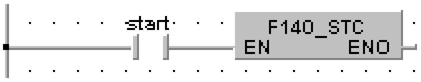


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function; result after a leading edge from start: carry-flag (R9009) will be set ON

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F140_STC();
END_IF;
```

F141_CLC Carry–flag reset

Steps 1

Description Special internal relay R9009 (carry–flag) goes OFF if the trigger **EN** is in the ON–state. This instruction can be used to control data using carry–flag R9009, e.g. F122_RCR and F123_RCL instructions.

PLC types

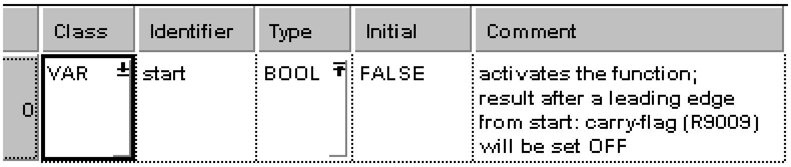
Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F141	x	x	x	x	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

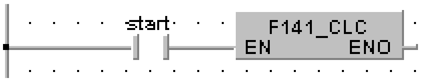


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function; result after a leading edge from start: carry-flag (R9009) will be set OFF

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F141_CLC();
END_IF;
```

F143_IORF

Partial I/O update

Steps **5**

Description

The instruction **F143_IORF** updates the inputs and outputs specified by **d1** (starting word address) and **d2** (ending word address) immediately after the trigger turns ON even in the program execution stage.

☞

- **With the FP0 or FP–Sigma, refreshing initiated by the IORF command is done only for the control unit.**
- **If d1 and d2 are variables and not constants, then the compiler automatically accesses the variables' values via the index register.**
- **With input refreshing, WX0 should be specified for d1 and d2.**
- **With output refreshing, WY0 should be specified for d1 and d2.**

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F143	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
d1	INT, WORD	starting word address
d2	INT, WORD	ending word address

The same type of operand should be specified for **d1** and **d2**.

Operands

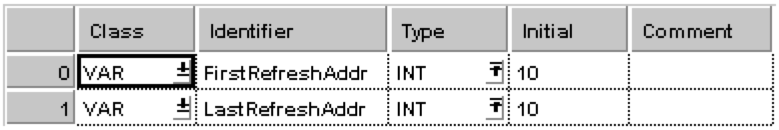
For	Relay				T/C		Register			Constant
	$WX_{(1)}$	$WY_{(1)}$	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
d1	x	x	–	–	–	–	–	–	–	–
d2	x	x	–	–	–	–	–	–	–	–

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

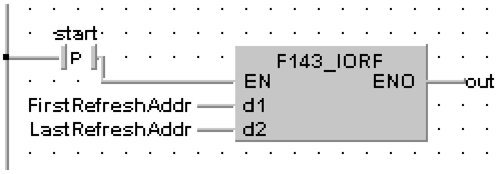


	Class	Identifier	Type	Initial	Comment
0	VAR	FirstRefreshAddr	INT	10	
1	VAR	LastRefreshAddr	INT	10	

Body

When the variable *start* changes from FALSE to TRUE, the function is carried out. To update WX10 and WY10 based on the master I/O map configuration, set **d1** = 10 and **d2** = 10.

LD



ST

```
(* PLCs without backplanes FP–M/FP–1/FP0/FP–Sigma *)
IF start THEN
      (* Updates the input/output relay of word no. 0 to 1
*)
      F143_IORF(WX0, WX1);
      F143_IORF(WY0, WY1);
END_IF;
```

If variables are used for the inputs d1 and d2 then FPWIN Pro internally uses index registers.

F148_ERR Self–diagnostic error set

Steps	3

Description

The error No. specified by **n*** is placed into special data register DT9000 (DT90000 for FP10/10S). At the same time, the self–diagnostic error–flag R9000 is set and ERROR LED on the CPU is turned ON. The contents of the error–flag R9000 can be read and checked using Control FPWIN Pro (**Monitor** → **Display Special Relays** → **Error Flag**). The error No., special data register DT9000 (DT90000 for FP10/10S), can be read and checked using Control FPWIN Pro (**Monitor** → **Display Special Registers** → **Basic Error Messages**). When **n*** = 0, the error is reset. (only for operation continue errors, **n*** = 200 to 299.) The ERROR LED is turned OFF and the contents of special data register DT9000 (DT90000 for FP10/10S) are cleared with 0. When **n*** = 100 to 199, the operation is halted. When **n*** = 200 to 299, the operation is continued.
Flag condition:

- Error–flag (R9007): Turns ON and keeps the ON state when the **n** exceeds the limit.
- Error–flag (R9008): Turns ON for an instant when the **n** exceeds the limit.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F148	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
n*	constant	self–diagnostic error code number, range: 0 and 100 to 299

Operands

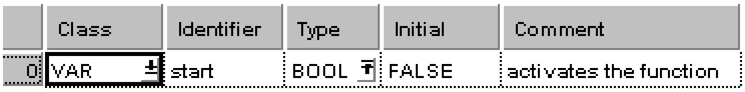
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	dec. or hex.
n*	–	–	–	–	–	–	–	–	–	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

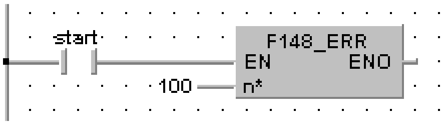


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      (* Sets the self-diagnostic error 100 *)
      (* The ERROR/ALARM LED of the PLC is on,
      and operation stops. *)
      F148_ERR(100);
END_IF;
```

F149_MSG

Message display

Steps	13

Description This instruction is used for displaying the message on the FP Programmer II screen. After executing **F149_MSG** instruction, you can see the message specified by **s** on the FP Programmer II screen. When the **F149_MSG** instruction is executed, the message–flag R9026 is set and the message specified by **s** is set in special data registers DT9030 to DT9035/DT90030 to DT90035. Once the message is set in special data registers, the message cannot be changed even if the **F149_MSG** instruction is executed again. You can clear the message with the FP Programmer II.

PLC types

Availability	FP0	FP1		FP–M	
	2.7k, 5k, 10k	0.9k	2.7k, 5k	0.9k	2.7k, 5k
F149	x	x	x	x	x

x: available
–: not available

Data types

Variable	Data type	Function
s	STRING(12)	message to be displayed

Operands

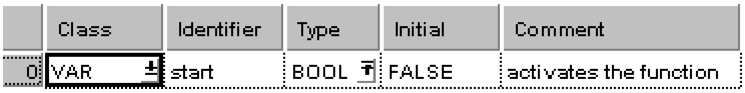
For	Relay				T/C		Register			Constant
	WX	WY	WR	WL	SV	EV	DT	LD	FL	character
s	–	–	–	–	–	–	–	–	–	x

x: available
–: not available

Example In this example the function is programmed in ladder diagram (LD) and structured text (ST). The same POU header is used for both programming languages. You can find an instruction list (IL) example in the online help.

POU header

In the POU header, all input and output variables are declared that are used for programming this function.

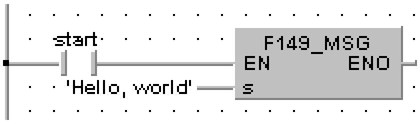


	Class	Identifier	Type	Initial	Comment
0	VAR	start	BOOL	FALSE	activates the function

Body

When the variable *start* is set to TRUE, the function is executed.

LD



ST

```
IF start THEN
      F149_MSG('Hello, world');
END_IF;
```

Appendix A

High–Speed Counter, Pulse and PWM Output

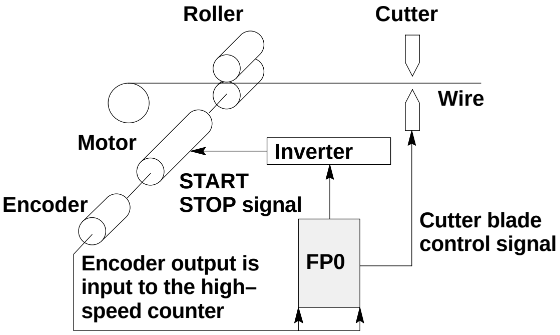
A.1 High–Speed Counter, Pulse and PWM Output

There are three functions available when using the high–speed counter built into the FP0 programmable controller. There are four channels for the built–in high–speed counter. The channel number allocated for the high–speed counter will change depending on the function being used.

The counting range is: K–8388608 to K8388607 (HFF8000 to H7FFFFF), coded 24–bit binary.

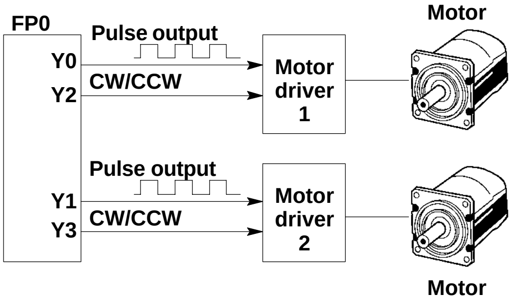
A.1.1 High–speed counter function

The high–speed counter function counts external inputs such as those from sensors or encoders. When the count reaches the target value, this function turns the desired output ON and OFF.



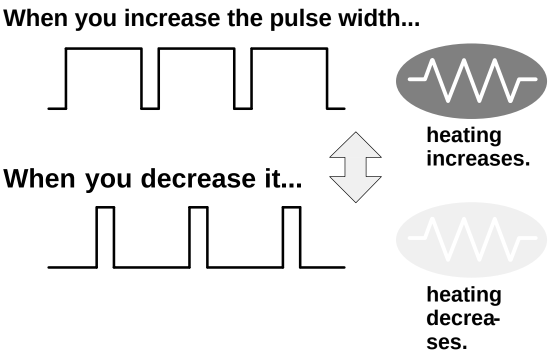
A.1.2 Pulse output function

Combined with a commercially available motor, the pulse output function enables positioning control. With the appropriate instruction, you can perform trapezoidal control, origin return, and JOG operation.



A.1.3 PWM output function

By using the appropriate instruction, the PWM output function enables a pulse output of the desired duty ratio.


When you increase the pulse width...
heating increases.
When you decrease it...
heating decreases.

A.2 Specifications and Restricted Items

A.2.1 Specifications

High–Speed Counter									
Input/output contact number being used			Built–in high–speed counter channel no.	Memory area used			Performance specifications		Related instructions
ON/OFF output	Count mode	Input contact number (value in parenthe–sis is reset input)		Contro l flag	Elapsed value area	Target value area	Min. input pulse width	Maximum counting speed	
Specify the desired output from Y0 to Y7	Incremental input Decrement–al input	**X0 (X2)**	CH0	R903A	DT9044 to DT9045	DT9046 to DT9047	50 ms <10 kHz>	Total of 4 CH with max. 10 kHz	**F0_MV** **F1_DMV** **F166_HC1S** **F167_HC1R**
		X1 (X2)	CH1	R903B	DT9048 to DT9049	DT905 0 to DT905 1			
		X3 (X5)	CH2	R903C	DT9104 to DT9105	DT910 6 to DT910 7	100 ms <5 kHz>		
		X4 (X5)	CH3	R903D	DT9108 to DT9109	DT911 0 to DT9111			
Specify the desired output from Y0 to Y7	2–phase input Incremental/ decremental input Directional distinction	**X0 X1 (X2)**	CH0	R903A	DT9044 to DT9045	DT904 6 to DT904 7	50 ms <10 kHz>	Total of 2 CH with max. 2 kHz	
		X3 X4 (X5)	CH2	R903C	DT9104 to DT9105	DT910 6 to DT910 7	100ms <5 kHz>		

Reset input X2 can be set to either CH0 or CH1. Reset input X5 can be set to either CH2 or CH3.

Pulse Output									
Input/output contact number being used				Built–in high–speed counter channel no.	Memory area used			Performance specifications for maximum output frequency	Related instructions
Pulse output	Direction–al output	Home input	Home proximity input		Control flag	Elapsed value area	Target value area		
Y0	**Y2**	**X0**	**DT9052 <bit2>**	CH0	R903A	DT9044 to DT9045	DT9046 to DT9047	Max. 10 kHz for 1–point output Max. 5 kHz for 2–point output	**F0_MV** **F1_DMV** **F168_SPD1** **F169_PLS**
Y1	**Y3**	**X1**	**DT9052 <bit6>**	CH1	R903B	DT9048 to DT9049	DT9050 to DT9051		

The maximum 1–point output for instruction F168 (SPD1) is 9.5 kHz.

PWM Output				
Output number being used	Built–in high–speed counter channel no.	Memory area used	Performance specifications for output frequency	Related instructions
		Control flag		
Y0	CH0	R903A	Frequency: 0.15 Hz to 38 Hz Duty: 0.1 % to 99.9 %	**F0_MV** **F1_DMV** **F170_PWM**
Y1	CH1	R903B		

A.2.2 Functions and Restrictions

The same channel cannot be used by more than one function, e.g. CH0 cannot be shared by the high–speed counter and pulse output functions.

The number allocated to each function cannot be used for normal input or outputs. Therefore the following examples are **NOT** possible:

- When using CH0 for 2–phase inputting with the high–speed counter function, you cannot allot X0 and X1 to normal inputs.
- When using Y0 for the pulse output function, you cannot allot origin input X0 to a normal input.
- When using Y0 for the pulse output (with directional output operating) function, you cannot allot Y2 (directional output) to a normal input or output.

When using the high–speed counter with a mode that does not use the reset input, you can allot the inputs listed in parenthesis in the specifications table to a normal input.

Example **When using the high–speed counter with no reset input and 2–phase input, you can allot X2 to a normal input.**

When any of the instructions related to the high–speed counter **(F166 to F170)** are executed, the control flag (special internal relay: R903A to R903D) corresponding to the used channel turns ON.

When the flag for a channel turns ON, another instruction cannot be executed using that same channel. For example, while executing **F166** (target value match ON instruction) and flag R903A is in the ON state, **F167** (target value match OFF instruction) **CANNOT** be executed with CH0.

The counting speed when using the high–speed counter function will differ depending on the counting mode as shown in the table. Therefore, the following restrictions apply:

- While in the incremental input mode and using the two channels CH0 and CH1, if CH0 is being used at 8 kHz, then CH1 can be used up to 2 kHz.
- While in the 2–phase input mode and using the two channels CH0 and CH2, if CH0 is being used at 1 kHz, then CH2 can be used up to 1 kHz.

The maximum output frequency when using the **pulse output function** will differ depending on the output contact number as shown in the table:

- When using either only Y0 or only Y1, the maximum output frequency is 10 kHz.
- When using the two contacts Y0 and Y1, the maximum output frequency is 5 kHz.

When using the high–speed counter function and pulse output function, specifications will differ depending on the conditions of use.

Example **When using one pulse output contact with a maximum output frequency of 5 kHz, the maximum counting speed of the high–speed counter being used simultaneously is 5 kHz with the incremental mode and 1 kHz with the 2–phase mode.**

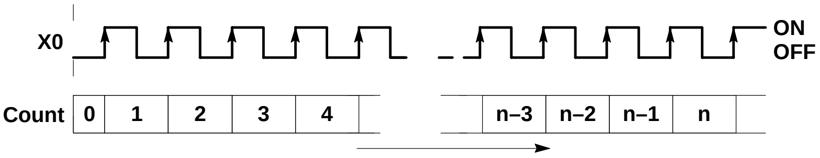
A.3 High–Speed Counter Function

- The high–speed counter function counts the input signals, and when the count reaches the target value, turns ON and OFF the desired output.
- The high–speed counter function is able to count high–speed pulses of frequencies up to 10 kHz.
- To turn ON an output when the target value is matched, use the target value match ON instruction F166. To turn OFF an output, use the target value match OFF instruction F167.
- Preset the output to be turned ON and OFF with the SET/RET instruction.

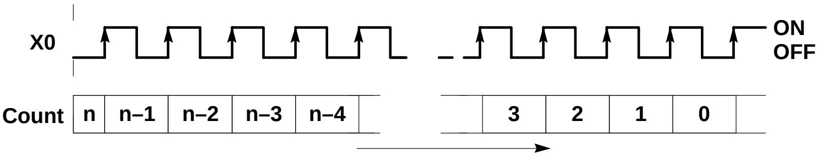
In order to use the high–speed counter function, it is necessary to set system registers No. 400 and No. 401.

A.3.1 Types of Input Modes

Incremental input mode:

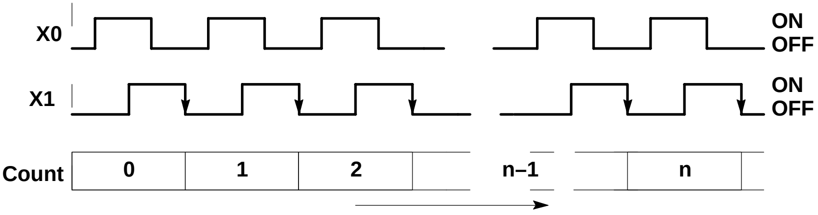


Decremental input mode:

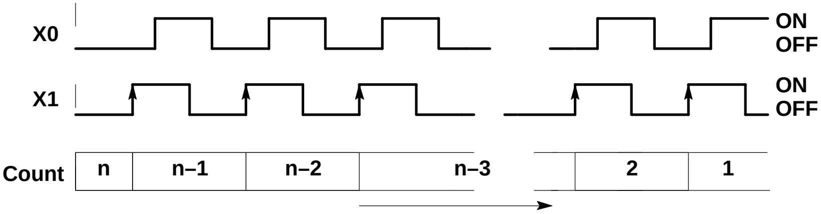


2–phase input mode:

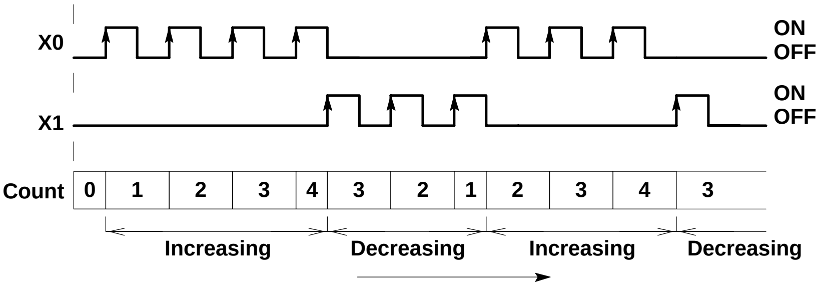
(Incremental input: CW)



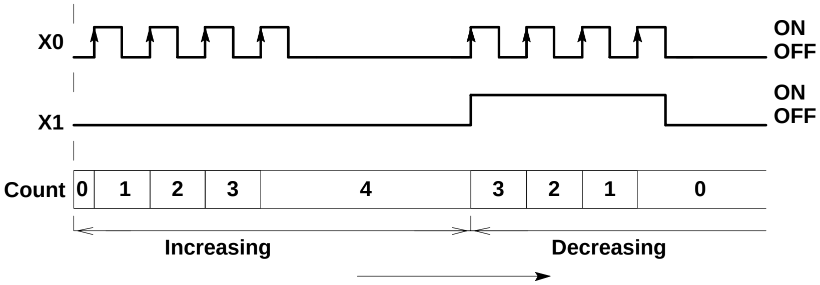
(Decremental input: CCW)



Incremental/decremental input mode (separate input mode):



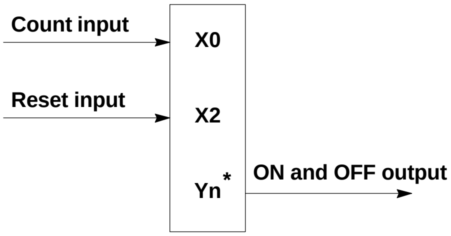
Directional distinction mode:



A.3.2 I/O Allocation

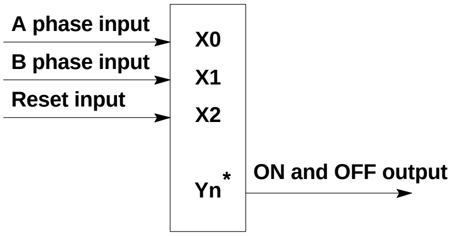
The input allocation, as shown in the table in section LEERER MERKER, will differ depending on the channel number being used. The output turned ON and OFF can be specified from between Y0 to Y7 as desired with instructions **F166** and **F167**.

Example 1: **When using CH0 with incremental input and reset input**



*** The output turned ON and OFF when values match can be selected from Y0 to Y7.**

Example 2: **When using CH0 with 2–phase input and reset input**



*** The output turned ON and OFF when values match can be selected from Y0 to Y7.**

A.4 Pulse Output Function

The pulse function enables positioning control by use in combination with a commercially available pulse–string input type motor driver. It provides trapezoidal control with the instruction **F168** for automatically obtaining pulse outputs by specifying the initial speed, maximum speed, acceleration/deceleration time, and target value. The **F168** instruction also enables automatic home return.

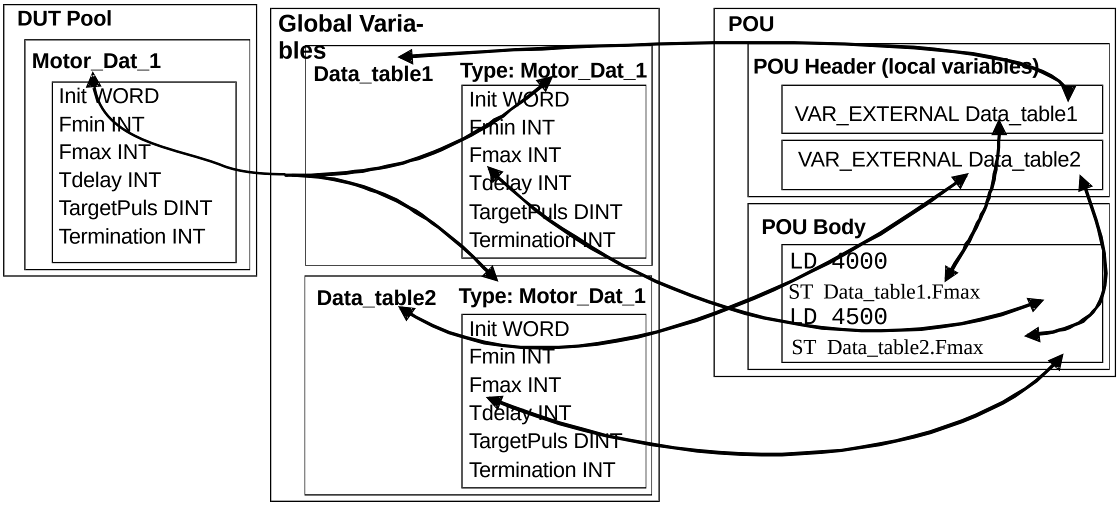
A JOG operation using instruction **F169** for pulse output while the predetermined trigger is in the ON state is also possible.

When using the pulse output function, set the channels corresponding to system registers No. 400 and No. 401 to “Do not use high–speed counter.”

A.4.1 SDT Variables

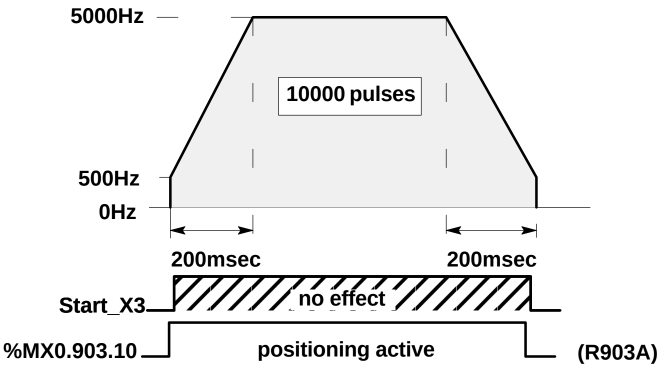
SDT Variables are used in the following example programs. SDT means Structured Data Type. These variables can be comprised of several kinds of variables (e.g. Word and Double Word).

SDT definitions or structures are administered globally and receive a structure name. For this structure, elements of various types are defined. If an SDT variable is to be used in a program, you need to assign an appropriate SDT variable in the global variable list. If one structure element of an SDT variable is to be accessed, the structure element must be separated from the structure variable name by a period (e.g. Data_table1.Fmax).

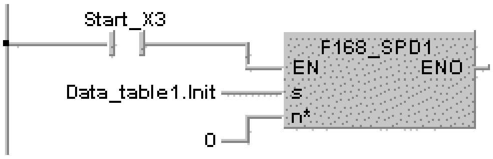


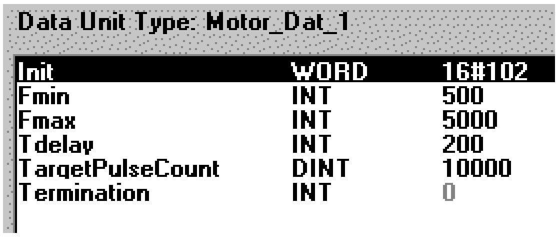
A.4.2 Positioning Function F168

This example illustrates normal positioning with an acceleration and a deceleration ramp.



The following program generates a pulse from output Y0. The initial speed is 500Hz, and the normal processing speed is 5000Hz. The acceleration and deceleration times are 200ms each. The movement amount is 10000 pulses.





Data Unit Type: Motor_Dat_1

Init	WORD	16#102
Fmin	INT	500
Fmax	INT	5000
Tdelay	INT	200
TargetPulseCount	DINT	10000
Termination	INT	0

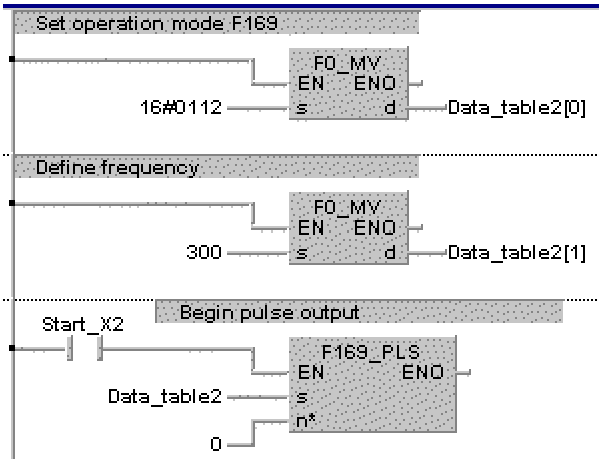
- **For trapezoidal control the initial and final speeds may not be greater than 5000Hz.**
- **The sum of maximum frequencies of all axes must not exceed 10000Hz.**

A.4.3 Pulse Output Function F169

The following example shows this process in a positive direction. The mode (of operation) 16#0112 sets the following conditions:

- The duty ratio is 10% pulse and 90% pause
- Incremental counting
- Directional output %QX0.2 (Y2) to ”0”.

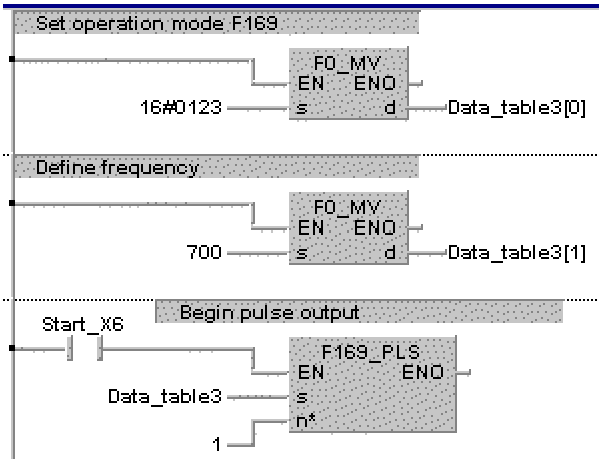
A frequency of 300Hz is output via the input Start_X2. During frequency output, the count of the elapsed value for the high–speed counter CH0 system registers (%MW0.904.8 and %MW0.904.9 (DT9048 u. DT9049), or %MW0.9004.8 and %MW0.9004.9 with the FP0–T32CP) decreases.



The following example shows this process in a negative direction. The mode (of operation) 16#0113 sets the following conditions:

- The duty ratio is 10% pulse and 90% pause
- Decremental counting
- Directional output %QX0.2 (Y2) to ”1”.

A frequency of 700Hz is output via the input Start_X6. During frequency output, the count of the elapsed value for the high–speed counter CH0 system registers (%MW0.904.8 and %MW0.904.9 (DT9048 u. DT9049), or %MW0.9004.8 and %MW0.9004.9 with the FP0–T32CP) decreases.



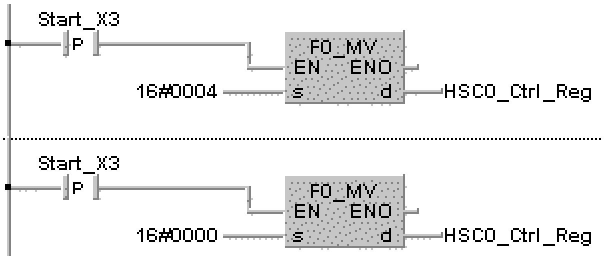
A.4.4 High–Speed Counter Control Instruction F0_MV

The function F0_MV is used for two different tasks. F0_MV is known as a MOVE function that copies values and memory contents. In addition, F0_MV is used to control the high–speed counter (e.g. for positioning a stepping motor). In this respect, F0_MV offers the following functionality:

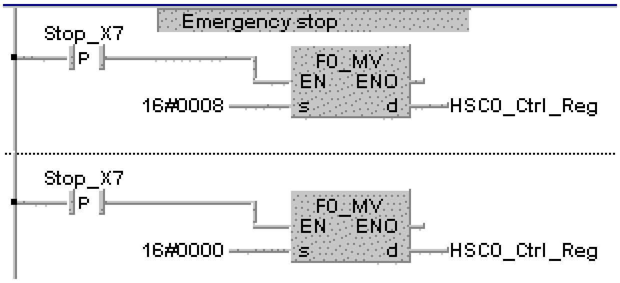
- This instruction is used for resetting the built–in high–speed counter, stopping the pulse outputs, and setting and resetting the home proximity input.

- Specify this instruction together with special data register %MW0.905.2 (DT9052) or %MW0.9005.2 with the FP0–T32CP.
- Once this instruction is executed, the settings will be retained until this instruction is executed again.

Example 1: **The home proximity speed is the starting speed of the ramp. The switching is enabled by assigning the value 4 to the high–speed counter special register (%MW0.905.2 (DT9052) or %MW0.9005.2 with the FP0–T32CP). ”0” is entered just after that to perform the preset operations.**



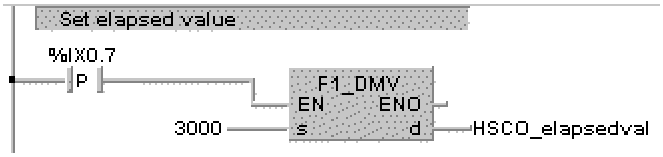
Example 2:



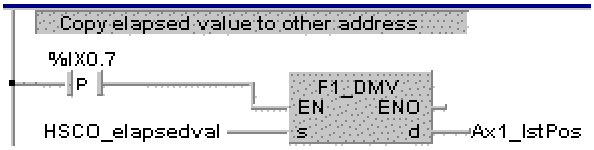
A.4.5 Elapsed Value Change and Read Instruction F1_DMV

In these examples, *HSCO_elapsedval* is assigned to the address %MD0.904.4 (DDT9044) or %MD0.9004.4 with the FP0–T32CP.

Example 1:

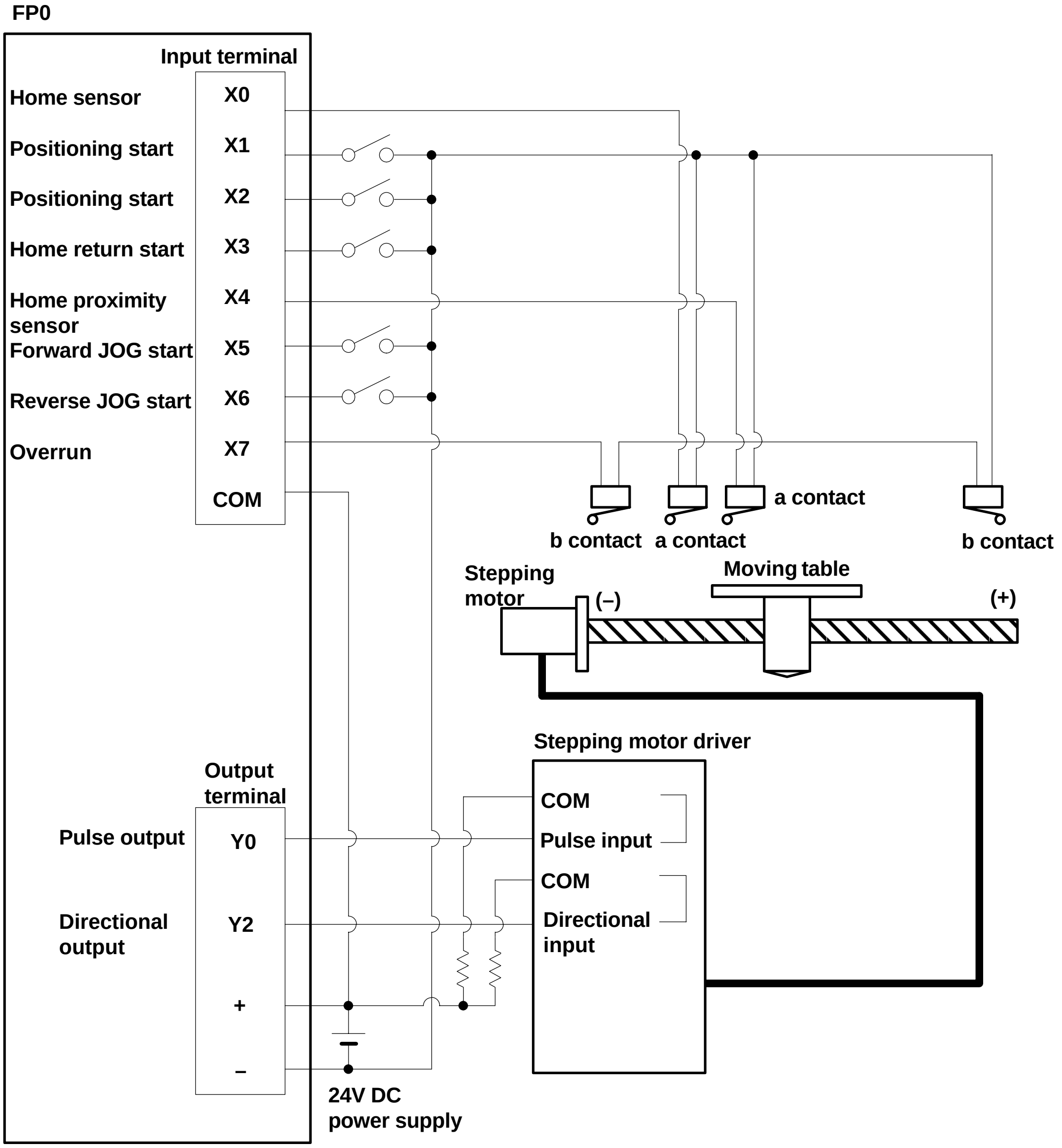


Example 2:



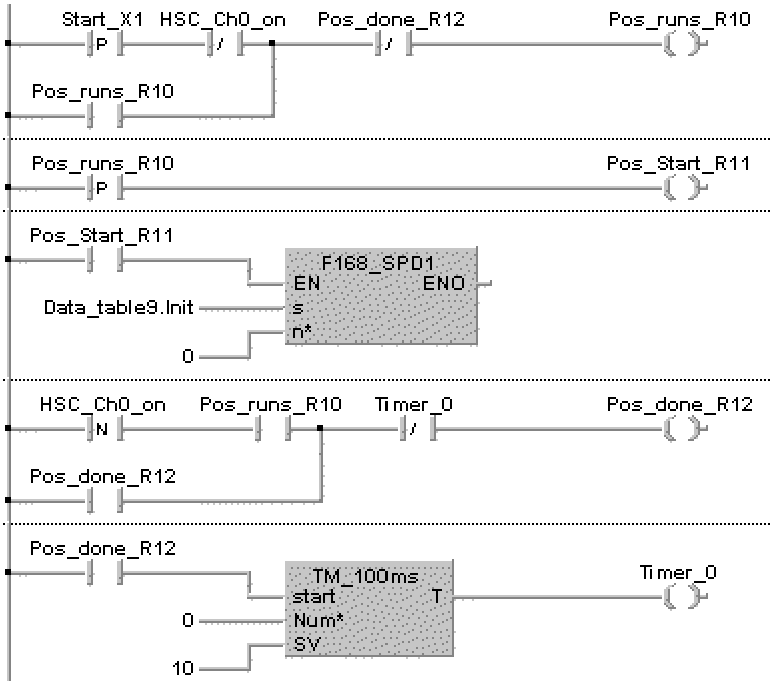
A.5 Sample Program for Positioning Control

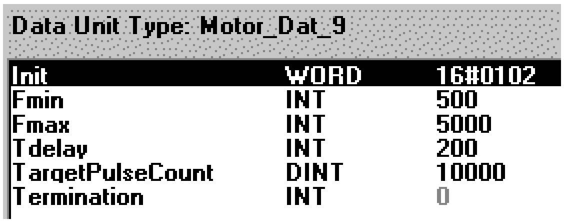
Wiring example



A.5.1 Relative Value Positioning Operation (Plus Direction)

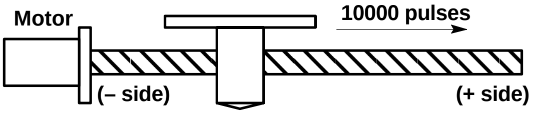
With **Start_X1** positioning starts. **Pos_runs_R10** indicates active positioning. Reaching the target position is indicated by **Pos_done_R12** for 1s.

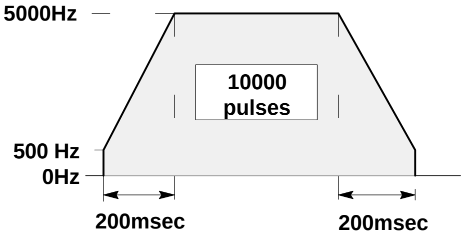




Data Unit Type: Motor_Dat_9

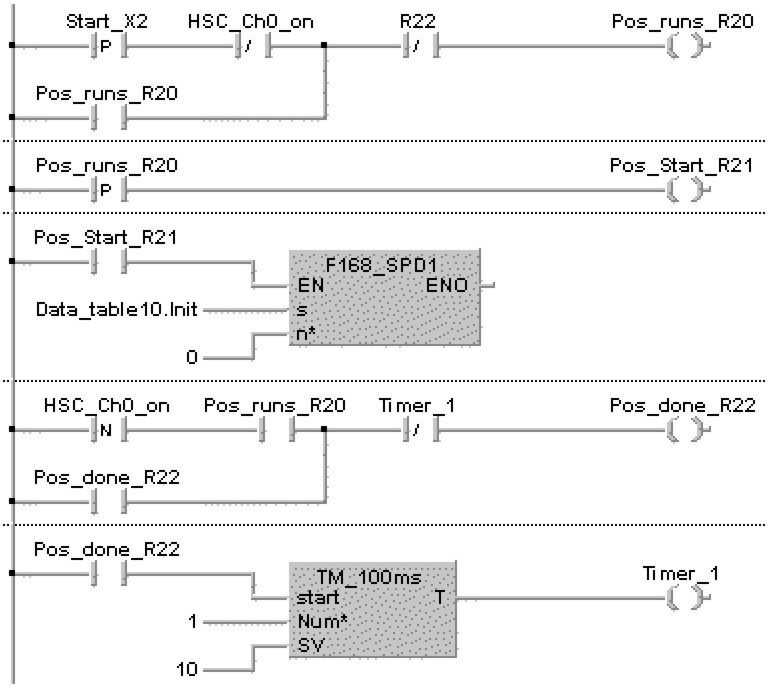
Init	WORD	16#0102
Fmin	INT	500
Fmax	INT	5000
Tdelay	INT	200
TargetPulseCount	DINT	10000
Termination	INT	0

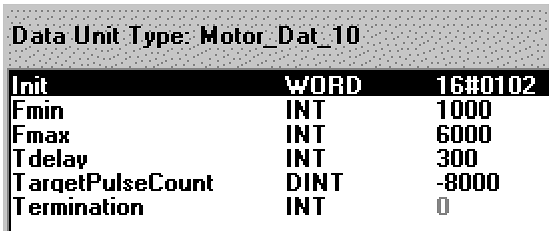




A.5.2 Relative Value Positioning Operation (Minus Direction)

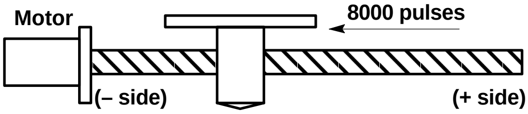
With **Start_X2** positioning starts. **Pos_runs_R20** indicates active positioning. Reaching the target position is indicated by **Pos_done_R22** for 1s.

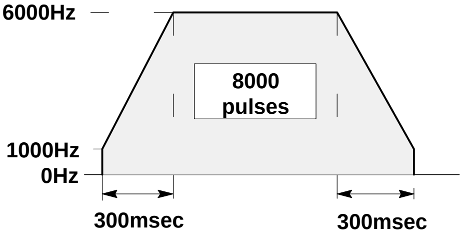




Data Unit Type: Motor_Dat_10

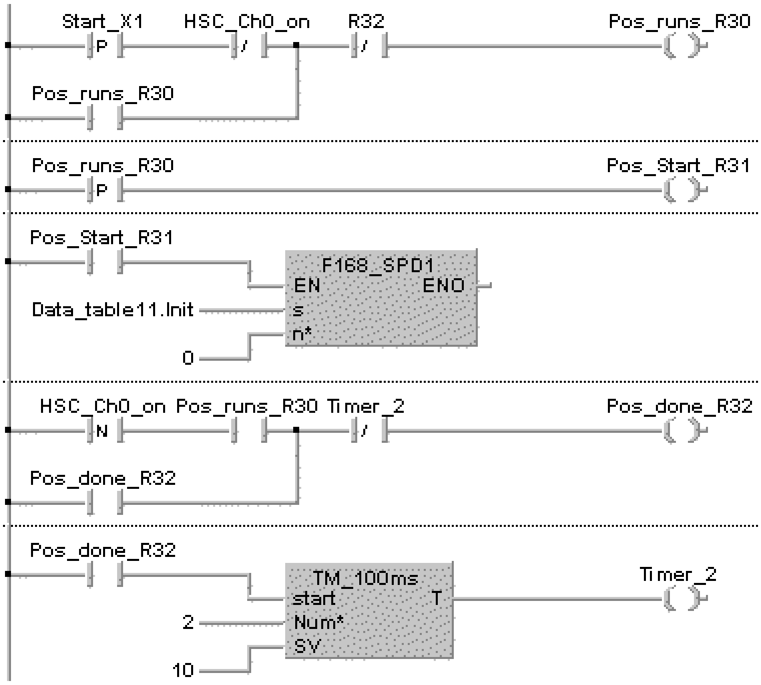
Init	WORD	16#0102
Fmin	INT	1000
Fmax	INT	6000
Tdelay	INT	300
TargetPulseCount	DINT	-8000
Termination	INT	0

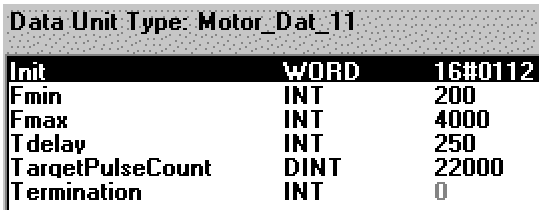




A.5.3 Absolute Value Positioning Operation

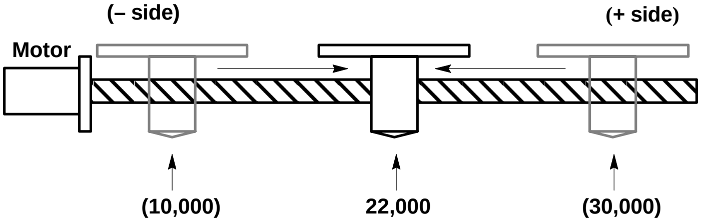
With **Start_X1** positioning starts. **Pos_runs_R30** indicates active positioning. Reaching the target position is indicated by **Pos_done_R32** for 1s. With absolute positioning, the directional output is controlled. The mode of operation 16#112 sets the directional output to ”1” when moving backward, and to ”0” when moving forward.

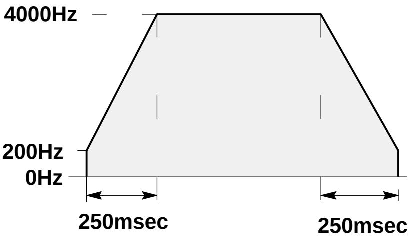




Data Unit Type: Motor_Dat_11

Init	WORD	16#0112
Fmin	INT	200
Fmax	INT	4000
Tdelay	INT	250
TargetPulseCount	DINT	22000
Termination	INT	0

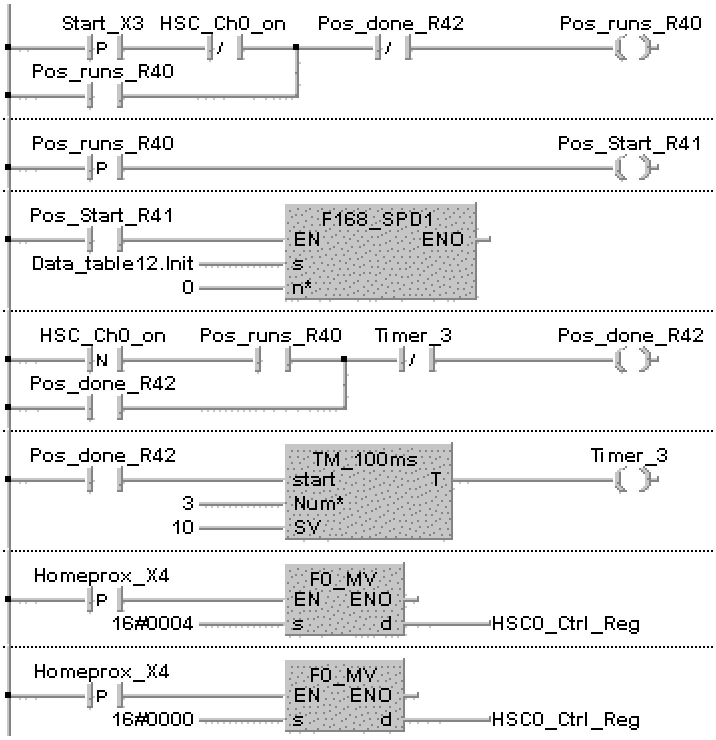


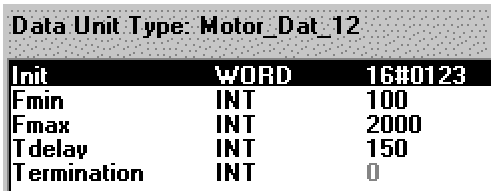


A.5.4 Home Return Operation (Minus Direction)

The return home direction causes the stepping motor to move in a reverse (minus) direction. The ramps are maintained, just as they are with other positioning processes. The braking ramp engages when the home proximity sensor turns on. Then the stepping motor runs at starting speed until the home sensor is activated. Then the pulse output stops, and the elapsed value is set to 0.

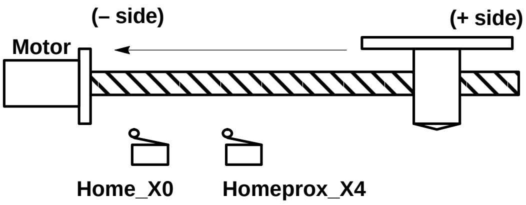
With **Start_X3** positioning starts. **Pos_runs_R40** indicates active positioning. **Pos_done_R42** turns on for 1s after the return home is completed, and the elapsed value (Addr. %MW0.904.4 and %MW0.904.5 (DT9044 and DT9045) or %MW0.9004.4 and %MW0.9004.5 with the FP0–T32CP) is set to 0.

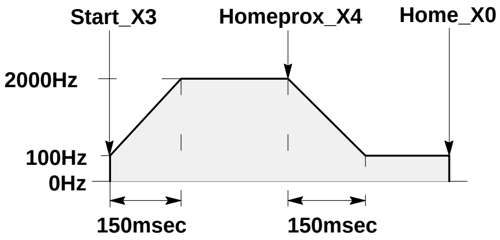




Data Unit Type: Motor_Dat_12

Init	WORD	16#0123
Fmin	INT	100
Fmax	INT	2000
Tdelay	INT	150
Termination	INT	0

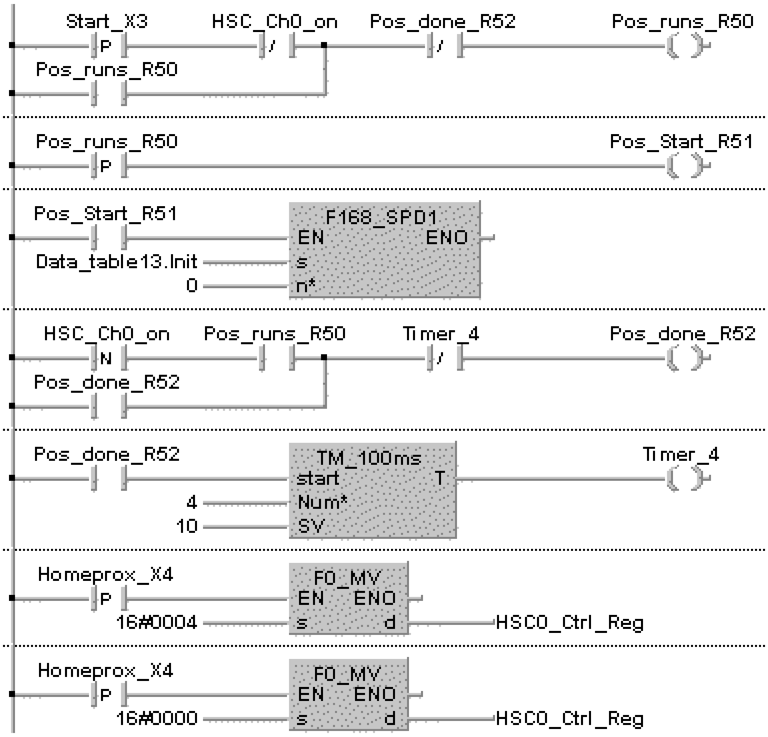


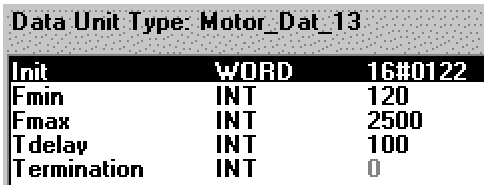


A.5.5 Home Return Operation (Plus Direction)

The return home direction causes the stepping motor to move in a forward (positive) direction. The ramps are maintained, just as they are with other positioning processes. The braking ramp engages when the home proximity sensor turns on. Then the stepping motor runs at starting speed until the home sensor is activated. Finally the pulse output stops, and the elapsed value is set to 0.

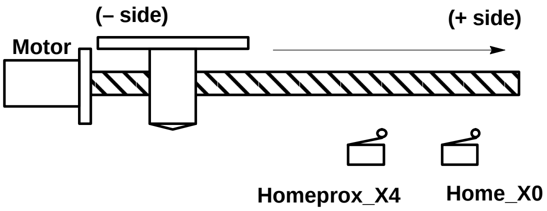
With **Start_X3** positioning starts. **Pos_runs_R50** indicates active positioning. **Pos_done_R52** turns on for 1s after the return home is completed, and the elapsed value (Addr. %MW0.904.4 and %MW0.904.5 (DT9044 and DT9045) or %MW0.9004.4 and %MW0.9004.5 with the FP0–T32CP) is set to 0.

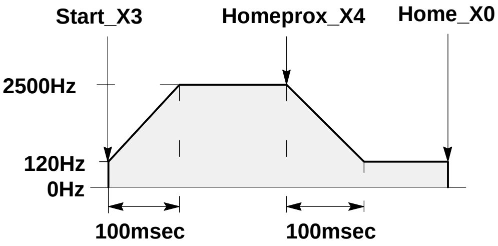




Data Unit Type: Motor_Dat_13

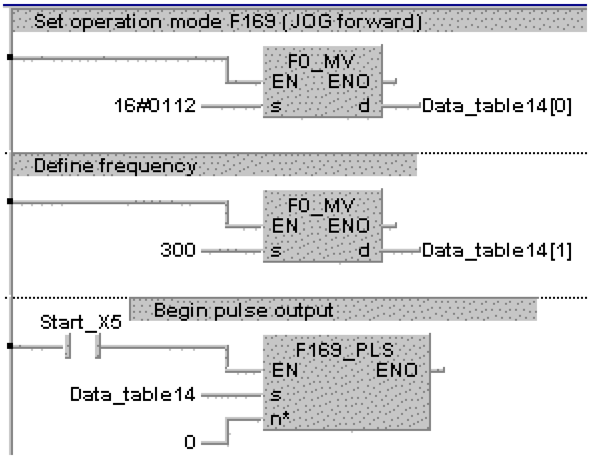
Init	WORD	16#0122
Fmin	INT	120
Fmax	INT	2500
Tdelay	INT	100
Termination	INT	0



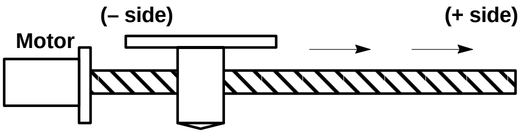


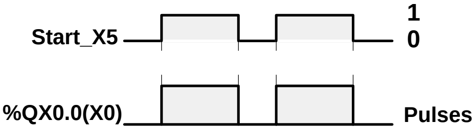
A.5.6 JOG Operation (Plus Direction)

The input Start_X5 starts the pulse output. The directional output %QX0.2 (Y2) is not controlled using this mode of operation (16#112).



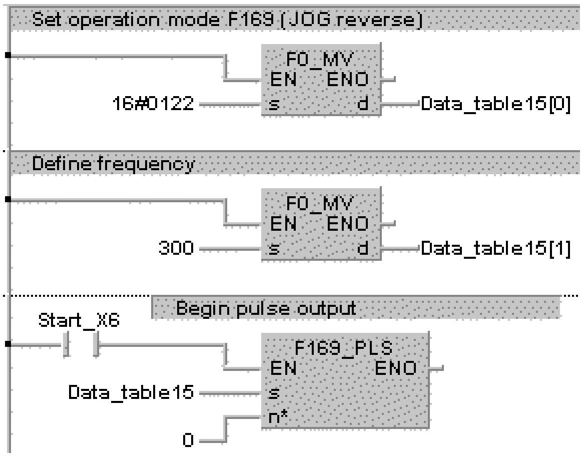
```
Set operation mode F169 (JOG forward)
                 F0_MV
                 EN   ENO
16#0112 ------   s      d ------ Data_table14[0]

Define frequency
                 F0_MV
                 EN   ENO
300 ----------   s      d ------ Data_table14[1]

Begin pulse output
Start_X5
--| |-------     F169_PLS
                 EN       ENO
Data_table14 --  s
                 n*
0 -------
```



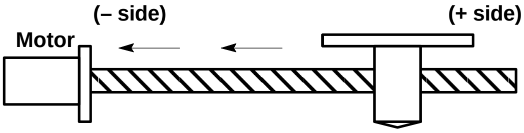


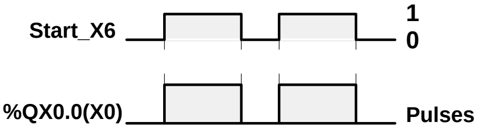
A.5.7 JOG Operation (Minus Direction)

The input Start_X6 starts the pulse output. The directional output %QX0.2 (Y2) is set using this mode of operation (16#122).



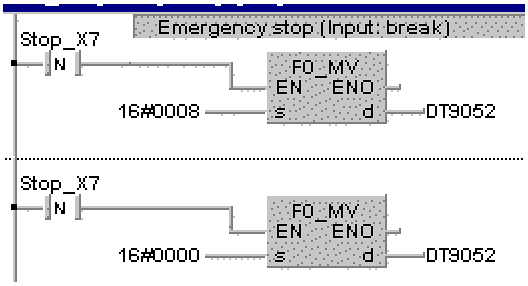
```
Set operation mode F169 (JOG reverse)
                 F0_MV
                 EN   ENO
16#0122 ------   s      d ------ Data_table15[0]

Define frequency
                 F0_MV
                 EN   ENO
300 ----------   s      d ------ Data_table15[1]

Begin pulse output
Start_X6
--| |-------     F169_PLS
                 EN       ENO
Data_table15 --  s
                 n*
0 -------
```





A.5.8 Emergency Stop

With a falling edge at the input, the pulse output is stopped. A break circuit has to be used as a protective circuit for this program. By using a break circuit, the emergency stop function is made fail–safe.



Appendix B

Special Data Registers

B.1 Special Data Registers FP0

The special data registers are one word (16-bit) memory areas which store specific information. With the exception of registers for which “Writing is possible” is indicated in the “Description” column, these registers cannot be written to.

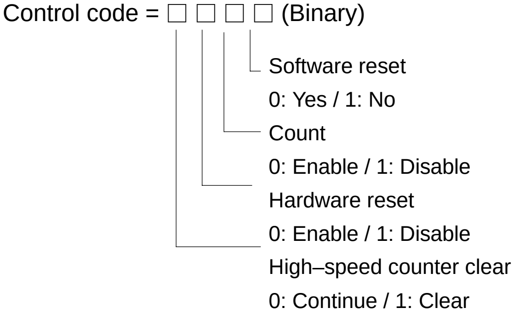
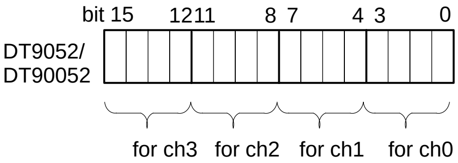
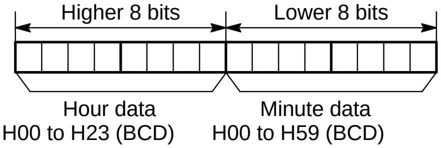
Address		**Name**	**Description**
FP0 T32	**FP0 C10, C14, C16, C32**		
DT90000	**DT9000**	**Self–diagnostic error code**	The self-diagnostic error code is stored here when a self-diagnostic error occurs. Monitor the error code using decimal display. For detailed information, section 8.6.3.
DT90010	**DT9010**	**I/O verify error unit**	The position of the I/O for which an error occurred is stored in bits 0 to 3.
DT90014	**DT9014**	**Auxiliary register for operation**	One shift-out hexadecimal digit is stored in bit positions 0 to 3 when **F105 (BSR)** or **F106 (BSL)** instruction is executed.
DT90015	**DT9015**	**Auxiliary register for operation**	The divided remainder (16-bit) is stored in DT9015/DT90015 when **F32 (%)** or **F52 (B%)** instruction is executed.
DT90016	**DT9016**		The divided remainder (32-bit) is stored DT9015 and DT9016/DT90015 and DT90016 when **F33 (D%)** or **F53 (DB%)** instruction is executed.
DT90017	**DT9017**	**Operation error address (hold)**	After commencing operation, the address where the first operation error occurred is stored. Monitor the address using decimal display.
DT90018	**DT9018**	**Operation error address (non-hold)**	The address where a operation error occurred is stored. Each time an error occurs, the new address overwrites the previous address. At the beginning of scan, the address is 0. Monitor the address using decimal display.
DT90019	**DT9019**	**2.5ms ring counter**	The data stored here is increased by one every 2.5ms. (H0 to HFFFF) Difference between the values of the two points (absolute value) X 2.5ms = Elapsed time between the two points.
DT90020	**DT9020**	————	Not used
DT90021	**DT9021**		
DT90022	**DT9022**	**Scan time (current value)** **(* Note)**	The current scan time is stored here. Scan time is calculated using the formula: Scan time (ms) = stored data (decimal) X 0.1 K50 indicates 5ms.

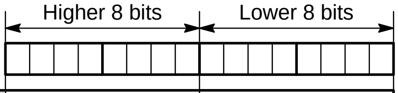
☞ **Scan time display is only possible in RUN mode, and shows the operation cycle time. The maximum and minimum values are cleared when each the mode is switched between RUN mode and PROG. mode.**

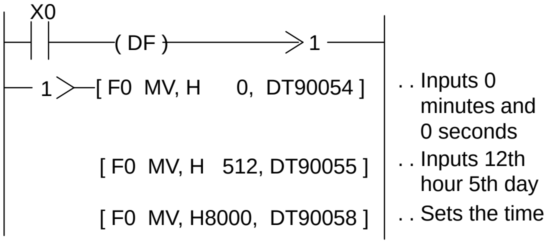
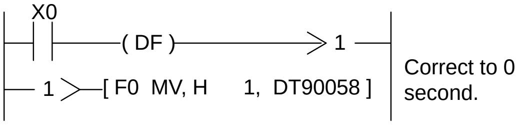
Address		Name	Description
FP0 T32	FP0 C10, C14, C16, C32		
DT90023	**DT9023**	**Scan time (minimum value) (* Note 1)**	The minimum scan time is stored here. Scan time is calculated using the formula: Scan time (ms) = stored data (decimal) X 0.1 K50 indicates 5ms.
DT90024	**DT9024**	**Scan time (maximum value) (* Note 1)**	The maximum scan time is stored here. Scan time is calculated using the formula: Scan time (ms) = stored data (decimal) X 0.1 K125 indicates 12.5ms.
DT90025 (* Note 2)	**DT9025 (* Note 2)**	**Mask condition monitoring register for interrupts (INT 0 to 5)**	The mask conditions of interrupts using **ICTL** instruction can be monitored here. Monitor using binary display. 15 11 7 3 0 (Bit No.) 23 19 16 (INT No.) 0: interrupt disabled (masked) 1: interrupt enabled (unmasked)
DT90026	**DT9026**	———————	Not used
DT90027 (* Note 2)	**DT9027 (* Note 2)**	**Periodical interrupt interval (INT 24)**	The value set by **ICTL** instruction is stored. – K0: periodical interrupt is not used – K1 to K3000: 10ms to 30s
DT90028	**DT9028**	———————	Not used
DT90029	**DT9029**	———————	Not used
DT90030 (* Note 2)	**DT9030 (* Note 2)**	**Message 0**	The contents of the specified message are stored in these special data registers when **F149 (MSG)** instruction is executed.
DT90031 (* Note 2)	**DT9031 (* Note 2)**	**Message 1**	
DT90032 (* Note 2)	**DT9032 (* Note 2)**	**Message 2**	
DT90033 (* Note 2)	**DT9033 (* Note 2)**	**Message 3**	
DT90034 (* Note 2)	**DT9034 (* Note 2)**	**Message 4**	
DT90035 (* Note 2)	**DT9035 (* Note 2)**	**Message 5**	
DT90036	**DT9036**	———————	Not used
DT90037	**DT9037**	**Work 1 for F96 (SRC) instruction**	The number of data that match the searched data is stored here when **F96 (SRC)** instruction is executed.

1) **Scan time display is only possible in RUN mode, and shows the operation cycle time. The maximum and minimum values are cleared when each the mode is switched between RUN mode and PROG. mode.**
2) **Used by the system.**

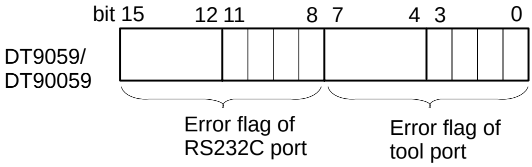
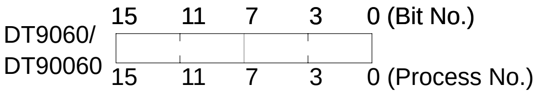
Address		Name	Description
FP0 T32	**FP0 C10, C14, C16, C32**		
DT90038	**DT9038**	**Work 2 for F96 (SRC) instruction**	The position of the first matching data, counting from the starting 16-bit area, is stored here when an **F96 (SRC)** instruction is executed.
DT90039 to DT90043	**DT9039 to DT9043**	————	Not used
DT90044	**DT9044**	**High-speed counter elapsed value for ch0**	The elapsed value (24–bit data) for the high–speed counter is stored here. Each time the **ED** instruction is executed, the elapsed value for the high–speed counter is automatically transferred to the special registers DT9044 and DT9045/DT90044 and DT90045. The value can be written by executing **F1 (DMV)** instruction.
DT90045	**DT9045**		
DT90046	**DT9046**	**High-speed counter target value for ch0**	The target value (24–bit data) of the high–speed counter specified by the high–speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high–speed counter related instruction **F166** to **F170** is executed. These preset values can only be read, and cannot be written.
DT90047	**DT9047**		
DT90048	**DT9048**	**High-speed counter elapsed value area for ch1**	The elapsed value (24–bit data) for the high–speed counter is stored here. Each time the **ED** instruction is executed, the elapsed value for the high–speed counter is automatically transferred to the special registers DT9048 and DT9049/DT90048 and DT90049. The value can be written by executing **F1 (DMV)**instruction.
DT90049	**DT9049**		
DT90050	**DT9050**	**High-speed counter target value area for ch1**	The target value (24–bit data) of the high–speed counter specified by the high–speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high–speed counter related instruction **F166** to **F170** is executed. These preset values can only be read, and cannot be written.
DT90051	**DT9051**		

Address		Name	Description
FP0 T32	FP0 C10, C14, C16, C32		
DT90052	**DT9052**	**High-speed counter control flag**	A value can be written with **F0 (MV)** instruction to reset the high-speed counter, disable counting, stop high-speed counter instruction **(F168)**, and clear the high-speed counter. Control code setting  Control code = □ □ □ □ (Binary) Software reset 0: Yes / 1: No Count 0: Enable / 1: Disable Hardware reset 0: Enable / 1: Disable High–speed counter clear 0: Continue / 1: Clear Software is not reset: H0 (0000) Perform software reset: H1 (0001) Disable count: H2 (0010) Disable hardware reset: H4 (0100) Stop pulse output (clear instruction): H8 (1000) Perform software reset and stop pulse output: H9 (1001) The 16 bits of DT9052/DT90052 are allocated in groups of four to high-speed channels 0 to 3 as shown below.  bit 15 1211 8 7 4 3 0 DT9052/ DT90052 for ch3 for ch2 for ch1 for ch0 A hardware reset disable is only effective when using the reset inputs (X2 and X5). In all other cases it is ignored. When using pulse output, a hardware reset input is equivalent to an home point proximate input.
DT90053	————	**Clock/calendar monitor (hour/minute)**	Hour and minute data of the clock/calendar are stored here. This data is read-only data; it cannot be overwritten.  Higher 8 bits Lower 8 bits Hour data H00 to H23 (BCD) Minute data H00 to H59 (BCD)

<table>
<tr><th colspan="2">Address</th><th rowspan="2">Name</th><th rowspan="2">Description</th></tr>
<tr><th>FP0 T32</th><th>FP0 C10, C14, C16, C32</th></tr>
<tr><td><b>DT90054</b></td><td>——</td><td><b>Clock/calendar monitor and setting (minute/second)</b></td><td rowspan="4">The year, month, day, hour, minute, second, and day-of-the-week data for the calendar timer is stored. The built-in calendar timer will operate correctly through the year 2099 and supports leap years. The calendar timer can be set (the time set) by writing a value using a programming tool software or a program that uses the <b>F0 (MV)</b> instruction.

<table>
<tr><td><b>DT90054</b></td><td>Minute data
H00 to H59 (BCD)</td><td>Second data
H00 to H59 (BCD)</td></tr>
<tr><td><b>DT90055</b></td><td>Day data
H01 to H31 (BCD)</td><td>Hour data
H00 to H23 (BCD)</td></tr>
<tr><td><b>DT90056</b></td><td>Year data
H00 to H99 (BCD)</td><td>Month data
H01 to H12 (BCD)</td></tr>
<tr><td><b>DT90057</b></td><td>——</td><td>Day-of-the-week data
H00 to H06 (BCD)</td></tr>
</table></td></tr>
<tr><td><b>DT90055</b></td><td>——</td><td><b>Clock/calendar monitor and setting (day/hour)</b></td></tr>
<tr><td><b>DT90056</b></td><td>——</td><td><b>Clock/calendar monitor and setting (year/month)</b></td></tr>
<tr><td><b>DT90057</b></td><td>——</td><td><b>Clock/calendar monitor and setting (day-of-the-week)</b></td></tr>
</table>

Address		Name	Description
FP0 T32	FP0 C10, C14, C16, C32		
DT90058	————	**Clock/calendar time setting and 30 seconds correction**	The clock/calendar is adjusted as follows. **When setting the clock/calendar by program** By setting the the highest bit of DT90058 to 1, the time becomes that written to DT90054 to DT90057 by **F0 (MV)** instruction. After the time is set, DT90058 is cleared to 0. (Cannot be performed with any instruction other than **F0 (MV)** instruction.) **Example:** Set the time to 12:00:00 on the 5th day when the X0 turns on.  X0 (DF) 1 1 [F0 MV, H 0, DT90054] . . Inputs 0 minutes and 0 seconds [F0 MV, H 512, DT90055] . . Inputs 12th hour 5th day [F0 MV, H8000, DT90058] . . Sets the time If you changed the values of DT90054 to DT90057 with the data monitor functions of programming tool software, the time will be set when the new values are written. Therefore, it is unnecessary to write to DT90058. **When the correcting times less than 30 seconds** By setting the lowest bit of DT90058 to 1, the value will be moved up or down and become exactly 0 seconds. After the correction is completed, DT90058 is cleared to 0. **Example:** Correct to 0 seconds with X0 turns on  X0 (DF) 1 1 [F0 MV, H 1, DT90058] Correct to 0 second. At the time of correction, if between 0 and 29 seconds, it will be moved down, and if the between 30 and 59 seconds, it will be moved up. In the example above, if the time was 5 minutes 29 seconds, it will become 5 minutes 0 second; and, if the time was 5 minutes 35 seconds, it will become 6 minutes 0 second.

After discharging the battery (including when the power is turned on for the first time), the values of DT90053 to DT90058 change at random. Once the time and date have been set, these values will function normally.

Address		Name		Description
FP0 T32	**FP0 C10, C14, C16, C32**			
DT90059	**DT9059**	**Serial communication error code**		 bit 15 12 11 8 7 4 3 0 DT9059/ DT90059 Error flag of RS232C port Error flag of tool port • Tool port bit 0 = 1: Over run error bit 1 = 1: Framing error bit 2 = 1: Parity error • RS232C port bit 8 = 1: Over run error bit 9 = 1: Framing error bit 10 = 1: Parity error
DT90060	**DT9060**	**Step ladder process**	**Process number: 0 to 15**	Indicates the startup condition of the step ladder process. When the proccess starts up, the bit corresponding to the process number turns on “1”. Monitor using binary display.  15 11 7 3 0 (Bit No.) DT9060/ DT90060 15 11 7 3 0 (Process No.) 0: not–executing 1: executing A programming tool software can be used to write data.
DT90061	**DT9061**		**Process number: 16 to 31**	
DT90062	**DT9062**		**Process number: 32 to 47**	
DT90063	**DT9063**		**Process number: 48 to 63**	
DT90064	**DT9064**		**Process number: 64 to 79**	
DT90065	**DT9065**		**Process number: 80 to 95**	
DT90066	**DT9066**		**Process number: 96 to 111**	
DT90067	**DT9067**		**Process number: 112 to 127**	
DT90104	**DT9104**	**High-speed counter elapsed value area for ch2**		The elapsed value (24–bit data) for the high–speed counter is stored here. Each time the **ED** instruction is executed, the elapsed value for the high–speed counter is automatically transferred to the special registers DT9104 and DT9105/DT90104 and DT90105. The value can be written by executing a **DMV (F1)** instruction.
DT90105	**DT9105**			
DT90106	**DT9106**	**High-speed counter target value area for ch2**		The target value (24–bit data) of the high–speed counter specified by the high–speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high–speed counter related instruction **F166** to **F170** is executed. These preset values can only be read, and cannot be written.
DT90107	**DT9107**			

Address		Name	Description
FP0 T32	FP0 C10, C14, C16, C32		
DT90108	**DT9108**	**High-speed counter elapsed value area for ch3**	The elapsed value (24–bit data) for the high–speed counter is stored here. Each time the **ED** instruction is executed, the elapsed value for the high–speed counter is automatically transferred to the special registers DT9108 and DT9109/DT90108 and DT90109. The value can be written by executing a **DMV (F1)** instruction.
DT90109	**DT9109**		
DT90110	**DT9110**	**High-speed counter target value area for ch3**	The target value (24–bit data) of the high–speed counter specified by the high–speed counter instruction is stored here. Target values have been preset for the various instructions, to be used when the high–speed counter related instruction **F166** to **F170** is executed. These preset values can only be read, and cannot be written.
DT90111	**DT9111**		

B.2 Special Data Registers FP–M/FP1

The special data registers are one word (16-bit) memory areas which store specific information. With the exception of registers for which “Writing is possible” is indicated in the “Description” column, these registers cannot be written to.

Address	Name	Description	Availability				
			FP1			FP-M	
			C14/ C16	C24/ C40	C56/ C72	C16	C20/ C32
DT9000	**Self-diagnostic error code register**	The self-diagnostic error code is stored in DT9000 when a self-diagnostic error occurs. Stores the error code using decimal number.	A	A	A	A	A
DT9014	**Auxiliary register for operation**	One shift-out hexadecimal digit is stored in hexadecimal digit position 0 (bit positions 0 to 3) when **F105 (BSR)** or **F106 (BSL)** instruction is executed.	A	A	A	A	A
DT9015	**Auxiliary register for operation**	The divided remainder (16–bit) is stored in DT9015 when **F32 (%)** or **F52 (B%)** instruction is executed.	A	A	A	A	A
DT9016		The divided remainder (32–bit) is stored in DT9015 and DT9016 when **F33 (D%)** or **F53 (DB%)** instruction is executed.	N/A	A	A	N/A	A
DT9017	**Operation error address (hold)**	After commencing operation, the address where the first operation error occurred is stored. Monitor the address using decimal display.	A (*)	A (*)	A (*)	A (*)	A (*)
DT9018	**Operation error address (non-hold)**	The address where a operation error occurred is stored. Each time an error occurs, the new address overwrites the previous address. At the beginning of scan, the address is 0. Monitor the address using decimal display.	A (*)	A (*)	A (*)	A (*)	A (*)
DT9019	**2.5ms ring counter register**	The data in DT9019 is increased by one every 2.5ms. Difference between the values of the two points (absolute value) X 2.5ms = Elapsed time between the two points.	A	A	A	A	A
DT9020	———	Not used	—	—	—	—	—
DT9021							

A: Available, N/A: Not available

☞ **Special data registers DT9017 and DT9018 are available for CPU version 2.7 or later.**

Address	Name	Description	Availability				
			FP1			FP-M	
			C14/ C16	C24/ C40	C56/ C72	C16	C20/ C32
DT9022	**Scan time (current value) (* Note 1)**	The current scan time is stored in DT9022. Scan time is calculated using the formula: Scan time (ms) = data X 0.1ms K50 indicates 5ms.	A	A	A	A	A
DT9023	**Scan time (minimum value) (* Note 1)**	The minimum scan time is stored in DT9023. Scan time is calculated using the formula: Scan time (ms) = data X 0.1ms K50 indicates 5ms.	A	A	A	A	A
DT9024	**Scan time (maximum value) (* Note 1)**	The maximum scan time is stored in DT9024. Scan time is calculated using the formula: Scan time (ms) = data X 0.1ms K125 indicates 12.5ms.	A	A	A	A	A
DT9025 (* Note 2)	**Mask condition monitoring register for interrupts (INT 0 to 7)**	The mask conditions of interrupts using **ICTL** instruction can be monitored here. Monitor using binary display. 15 11 7 3 0 (Bit No.) 23 19 16 (INT No.) 0: interrupt disabled (masked) 1: interrupt enabled (unmasked)	N/A	A	A	N/A	A
DT9026	——	Not used	——	——	——	——	——
DT9027 (* Note 2)	**Periodical interrupt interval (INT24)**	The value set by **ICTL** instruction is stored. – K0: periodical interrupt is not used – K1 to K3000: 10 ms to 30 s	N/A	A	A	N/A	A

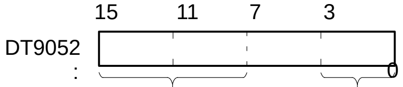
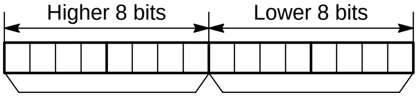
1) **The scan time display is during the RUN mode only and displays the operation cycle time. During the PROG. mode, the operation scan time is not displayed. The maximum and minimum values are cleared when each the mode is switched between the RUN and PRG. modes.**

2) **Used by the system.**

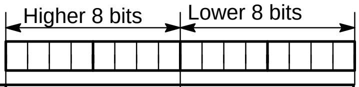
Address	Name	Description	Availability				
			FP1			FP-M	
			C14/ C16	C24/ C40	C56/ C72	C16	C20/ C32
DT9028	——	Not used	——	——	——	——	——
DT9029	——	Not used	——	——	——	——	——
DT9030 **(* Note)**	**Message 0**	The contents of the specified message are stored in DT9030, DT9031, DT9032, DT9033, DT9034, and DT9035 when **F149 (MSG)** instruction is executed.	N/A	A	A	N/A	A
DT9031 **(* Note)**	**Message 1**		N/A	A	A	N/A	A
DT9032 **(* Note)**	**Message 2**		N/A	A	A	N/A	A
DT9033 **(* Note)**	**Message 3**		N/A	A	A	N/A	A
DT9034 **(* Note)**	**Message 4**		N/A	A	A	N/A	A
DT9035 **(* Note)**	**Message 5**		N/A	A	A	N/A	A
DT9036	——	Not used	——	——	——	——	——
DT9037	**Work 1 for F96 (SRC) instruction**	The number of that match the searched data is stored in DT9037 when **F96 (SRC)** instruction is executed.	A	A	A	A	A
DT9038	**Work 2 for F96 (SRC) instruction**	The position of the first matching data, counting from the starting 16-bit area, is stored in DT9038 when **F96 (SRC)** instruction is executed.	A	A	A	A	A
DT9039	——	Not used	——	——	——	——	——
DT9040	**Manual dial–set register (V0)**	Stores the potentiometer input value (K0 to K255) – FP1 C14, 16: V0 → DT9040 – FP1 C24 and FP–M C20, C32: V0 → DT9040, V1 → DT9041 – FP–M C16: V0 → DT9040, V1 → DT9041 V2 → DT9042 – FP1 C40, C56, and C72: V0 → DT9040, V1 → DT9041 V2 → DT9042, V3 → DT9043	A	A	A	A	A
DT9041	**Manual dial–set register (V1)**		N/A	A	A	A	A
DT9042	**Manual dial–set register (V2)**		N/A	A (C40 only)	A	A	N/A
DT9043	**Manual dial–set register (V3)**		N/A	N/A	A	N/A	N/A
DT9044	**High–speed counter elapsed value for built–in high–speed counter**	The high-speed counter elapsed value (24 bits data) is stored in DT9044 and DT9045. The value can be written by executing **F1 (DMV)** instruction.	A	A	A	A	A
DT9045			A	A	A	A	A
DT9046	**High–speed counter target value for built–in high–speed counter**	The high-speed counter target value (24 bits data) specified by **F162 (HC0S)** to **F164 (SPD0)** instructions is stored in DT9046 and DT9047.	A	A	A	A	A
DT9047			A	A	A	A	A

A: Available, N/A: Not available

☞ **Used by the system.**

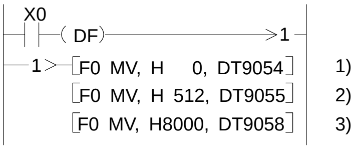
Address	Name	Description	Availability				
			FP1			FP-M	
			C14/ C16	C24/ C40	C56/ C72	C16	C20/ C32
DT9048 to DT9051	——	Not used	——	——	——	——	——
DT9052	**Built–in high–speed counter control flag**	A value can be written with **F0 (MV)** instruction to reset the high-speed counter, disable counting, stop high-speed counter instructions (**F162** to **F165**), and clear the high-speed counter. Control code = 000000000000□□□□ High–speed counter instruction (0: Continue / 1: Clear) Hardware reset (0: Enable / 1: Disable) Count (0: Enable / 1: Disable) Software reset (0: Yes / 1: No) The system register 400 setting is stored in the upper 16 bits.  <Mode setting> Set by system register 400 (H00 to H08) <Control code> Entered by the **F0 (MV)** instruction (H0 to HF)	A	A	A	A	A
DT9053	**Clock/calendar monitor (hour and minute)**	Hour and minute data of the clock/calendar are stored in DT9053. This data is read–only data; it cannot be overwritten.  Hour data H00 to H23 (BCD) Minute data H00 to H59 (BCD)	N/A	A (*)	A (*)	N/A	A (*)

C type FP–M C20, C32 and FP1 C24C, C40C, C56C, and C72C only.

<table>
<tr><th rowspan="3">Address</th><th rowspan="3">Name</th><th rowspan="3">Description</th><th colspan="5">Availability</th></tr>
<tr><th colspan="3">FP1</th><th colspan="2">FP-M</th></tr>
<tr><th>C14/ C16</th><th>C24/ C40</th><th>C56/ C72</th><th>C16</th><th>C20/ C32</th></tr>
<tr><td><b>DT9054</b></td><td><b>Clock/calendar monitor and setting (minute and second)</b></td><td rowspan="4">The year, month, day, hour, minute, second, and day-of-the-week data for the calendar timer is stored. The built-in calendar timer will operate correctly through the year 2099 and supports leap years. For CPU Ver. 2.1 or later, the calendar timer can be set (the time set) by writing a value using a programming tool software or a program that uses the <b>F0 (MV)</b> instruction.

<table>
<tr><td><b>DT9054</b></td><td>Minute data H00 to H59 (BCD)</td><td>Second data H00 to H59 (BCD)</td></tr>
<tr><td><b>DT9055</b></td><td>Day data H01 to H31 (BCD)</td><td>Hour data H00 to 23 (BCD)</td></tr>
<tr><td><b>DT9056</b></td><td>Year data H00 to H99 (BCD)</td><td>Month data H01 to H12 (BCD)</td></tr>
<tr><td><b>DT9057</b></td><td>————————</td><td>Day–of–the–week data H00 to H06 (BCD)</td></tr>
</table></td><td>N/A</td><td>A (*)</td><td>A (*)</td><td>N/A</td><td>A (*)</td></tr>
<tr><td><b>DT9055</b></td><td><b>Clock/calendar monitor and setting (day and hour)</b></td><td>N/A</td><td>A (*)</td><td>A (*)</td><td>N/A</td><td>A (*)</td></tr>
<tr><td><b>DT9056</b></td><td><b>Clock/calendar monitor and setting (year and month)</b></td><td>N/A</td><td>A (*)</td><td>A (*)</td><td>N/A</td><td>A (*)</td></tr>
<tr><td><b>DT9057</b></td><td><b>Clock/calendar monitor and setting (day–of–the–week)</b></td><td>N/A</td><td>A (*)</td><td>A (*)</td><td>N/A</td><td>A (*)</td></tr>
</table>

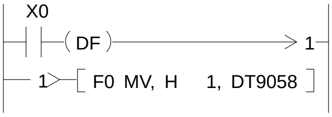
A: Available, N/A: Not available

C type FP–M C20, C32 and FP1 C24C, C40C, C56C, and C72C only.

Address	Name	Description	Availability				
			FP1			FP-M	
			C14/ C16	C24/ C40	C56/ C72	C16	C20/ C32
DT9058	**Clock/calendar time setting and 30 seconds correction**	The clock/calendar is adjusted as follows, • **When setting the clock/calendar by program (CPU version 2.1 or later)** By setting the highest bit of DT9058 to 1, the time becomes that written to DT9054 to DT9057 by F0(MV) instruction. After the time is set, DT9058 is cleared to 0. Example: Set to time 12:00:00 on the 5th day with X0 turns on.  X0 DF 1 1 F0 MV, H 0, DT9054 F0 MV, H 512, DT9055 F0 MV, H8000, DT9058 1) 2) 3) 1) Inputs 0 minutes and 0 seconds 2) Inputs 12th hour and 5th day 3) Sets the time If you changed the values of DT9054 to DT9057 with the data monitor functions of programming tool software, the time will be set when the new values are written. Therefore, it is unnecessary to write to DT9058.	N/A	A (*)	A (*)	N/A	A (*)

A: Available, N/A: Not available

C type FP–M C20, C32 and FP1 C24C, C40C, C56C, and C72C only.

Address	Name	Description	Availability				
			FP1			FP-M	
			C14/ C16	C24/ C40	C56/ C72	C16	C20/ C32
DT9058	**Clock/calendar time setting and 30 seconds correction**	**When the correcting times less than 30 seconds** By setting the lowest bit of DT9058 to 1, the value will be moved up or down and become exactly 0 seconds. After the correction is completed, DT9058 is cleared to 0. Example: Correct to 0 seconds with X0:on  Correct to 0 second. At the time of correction, if between 0 and 29 seconds, it will be moved down, and if between 30 and 59 seconds, it will be moved up. In the example above, if the time was 5 minutes 29 seconds, it will become 5 minutes 0 seconds; and, if the time was 5 minutes 35 seconds, it will become 6 minutes 0 seconds.	N/A	A (* 1)	A (* 1)	N/A	A (* 1)
DT9059 **(* Note 2)**	**Serial communication error code**	Higher 8–bit: Error code of RS232C port is stored. Lower 8–bit: Error code of tool port is stored.	A (* 1)	N/A	A (* 1)	N/A	A (* 1)

A: Available, N/A: Not available

1) C type FP–M C20, C32 and FP1 C24C, C40C, C56C, and C72C only.

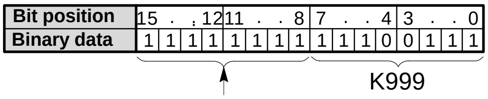
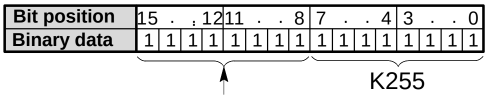
2) Used by the system.

Address	**Name**		**Description**	**Availability**				
				FP1			**FP-M**	
				C14/ C16	**C24/ C40**	**C56/ C72**	**C16**	**C20/ C32**
DT9060	**Step ladder process**	**Process number: 0 to 15**	Indicates the startup condition of the step ladder process. When the process starts up, the bit corresponding to the process number turns on “1”. Monitor using binary display. 15 11 7 3 0 (Bit No.) DT9060 15 11 7 3 0 (Process No.) 0: not–executing 1: executing A programming tool software can be used to write data.	A	A	A	A	A
DT9061		**Process number: 16 to 31**		A	A	A	A	A
DT9062		**Process number: 32 to 47**		A	A	A	A	A
DT9063		**Process number: 48 to 63**		A	A	A	A	A
DT9064		**Process number: 64 to 79**		N/A	A	A	A	A
DT9065		**Process number: 80 to 95**		N/A	A	A	A	A
DT9066		**Process number: 96 to 111**		N/A	A	A	A	A
DT9067		**Process number: 112 to 127**		N/A	A	A	A	A

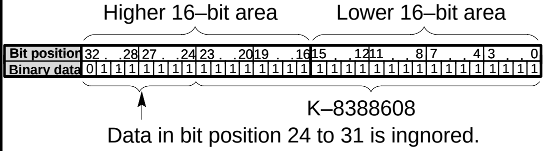
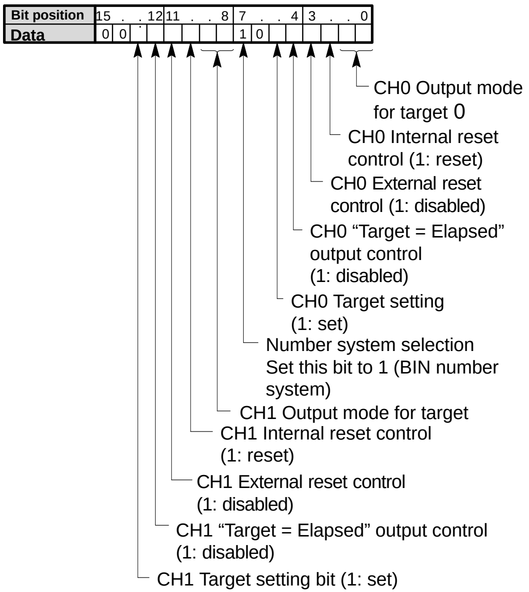
A: Available, N/A: Not available

Address	Name		Description	Availability		
				FP1	FP–M	
					C16	C20/ C32
DT9080	**Digital converted value from analog control board No.0**	**Channel 0**	These registers are used to store digital converted value of analog inputs from A/D converter board or analog I/O board. The range of digital converted value depends on the type of analog control boards as follows: **When A/D converter board is installed** K0 to K999 (0 to 20mA/0 to 5V/0 to 10V) Range of digital converted value (10 bits resolution) **If analog data over the maximum analog value (20mA/5V/10V) is input, digital data up to K1023 is available.** **However, be sure to input analog voltage or analog current within the rated range in order to prevent system damages.** **When Analog I/O board is installed** K0 to K255 (0 to 20mA/0 to 5V/0 to 10V) Range of digital converted value (6 bits resolution) **Even if analog data outside the specified range is input, digital converted value outside K0 to K255 is not available.** **Be sure to input analog voltage within the rated range in order to prevent system damages.** Be sure to use the **F0 (MV)** instruction to transfer data in these special data registers into other data registers.	N/A	N/A	A
DT9081		**Channel 1**				
DT9082		**Channel 2**				
DT9083		**Channel 3**				
DT9084	**Digital converted value from analog control board No.1**	**Channel 0**		N/A	N/A	A
DT9085		**Channel 1**				
DT9086		**Channel 2**				
DT9087		**Channel 3**				
DT9088	**Digital converted value from analog control board No.2**	**Channel 0**		N/A	N/A	A
DT9089		**Channel 1**				
DT9090		**Channel 2**				
DT9091		**Channel 3**				
DT9092	**Digital converted value from analog control board No.3**	**Channel 0**		N/A	N/A	A
DT9093		**Channel 1**				
DT9094		**Channel 2**				
DT9095		**Channel 3**				

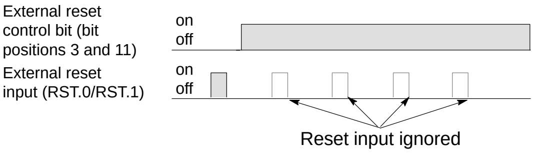
A: Available, N/A: Not available

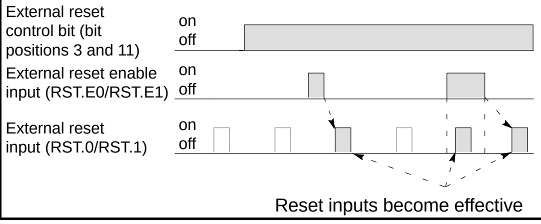
Address	Name		Description	Availability		
				FP1	FP–M	
					C16	C20/ C32
DT9096	**Digital value for specifying analog data output from analog control board No.0**	**Channel 0**	These registers are used to specify data for analog output from D/A converter boards or analog I/O boards. The range of digital value to specify analog output depends on the type of analog control boards as follows: **When D/A converter boards is installed** Range of deigital data for specifying analog output (10 bits): K0 to K999 (0 to 20mA/0 to 5V/0 to 10V) **Be sure to specify data within the range of K0 to K999.** **– If data K1000 to K1023 is specified, analog data a little bit more than the maximum rated value (20mA/5V/10V) is output.** **– If data outside K0 to K1023 is specified, data is handled disregarding data in bit positions 10 to 15.** **Example:** **If K–24 is input, analog data is output regarding it as K999. Data configuration when K–24 is input**  Bit position 15 . . 12 11 . . 8 7 . . 4 3 . . 0 Binary data 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 K999 Data in bit position 10 to 15 is ingnored. **When Analog I/O boards is installed** Range of deigital data for specifying analog output (6 bits): K0 to K255 (0 to 20mA/0 to 5V/0 to 10V) **Be sure to specifying data within the range of K0 to K255. If data outside K0 to K255 is specified, data is handled disregarding data in bit positions 6 to 15.** **Example:** **If K–1 is input, analog data is output regarding it as K255. Data configuration when K–1 is input**  Bit position 15 . . 12 11 . . 8 7 . . 4 3 . . 0 Binary data 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 K255 Data in bit position 8 to 15 is ingnored. Be sure to use the **F0 (MV)** instruction to transfer data into these special data registers.	N/A	N/A	A
DT9097		**Channel 1**				
DT9098	**Digital value for specifying analog data output from analog control board No.1**	**Channel 0**		N/A	N/A	A
DT9099		**Channel 1**				
DT9100	**Digital value for specifying analog data output from analog control board No.2**	**Channel 0**		N/A	N/A	A
DT9101		**Channel 1**				
DT9102	**Digital value for specifying analog data output from analog control board No.3**	**Channel 0**		N/A	N/A	A
DT9103		**Channel 1**				

A: Available, N/A: Not available

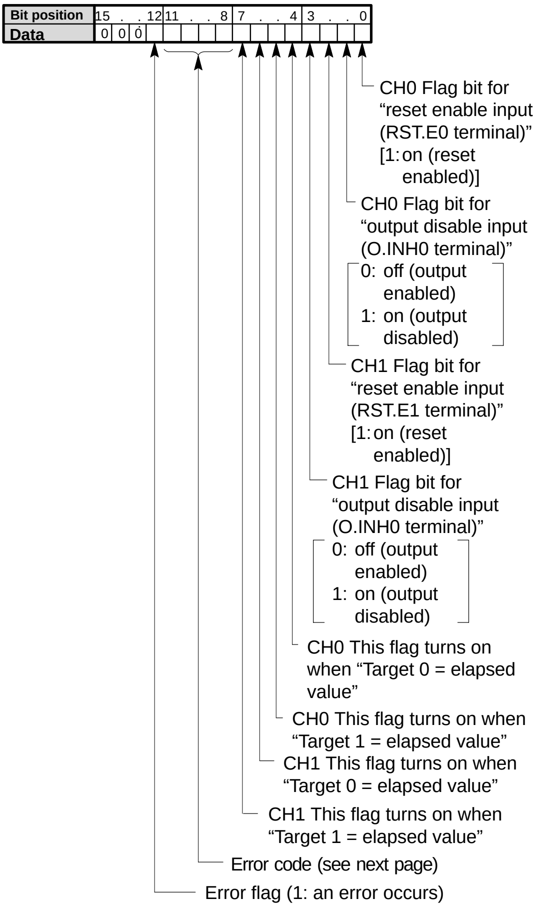
Address	Name		Description	Availability		
				FP1	FP–M	
					C16	C20/ C32
DT9104 DT9105	Target value 0 of high–speed counter board	Channel 0	These registers are performed for storing data of the FP–M high–speed counter board. The target values 0 and 1, elapsed value, and capture value are processed in binary in the range of K–8388608 to 8388607. • **Be sure to use F1 (DMV) instruction to transfer data in these special data registers to other registers or data in other registers to these special data registers.** • **When changing data in these special data registers, be sure to specify data in the range of K–8388608 to K8388607. If data outside range is input, data handled disregarding bit positions 24 to 31 (bit positions 8 to 15 in the higher 16–bit area of 32–bit data).** Example: If K2147483647 is specified, high–speed counter acts regarding it as K–8388608. Data configuration when K2147483647 is input:  Higher 16–bit area Lower 16–bit area Bit position 32 . . 28 27 . . 24 23 . . 20 19 . . 16 15 . . 12 11 . . 8 7 . . 4 3 . . 0 Binary data 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 K–8388608 Data in bit position 24 to 31 is ingnored.	N/A	N/A	A
DT9106 DT9107	Target value 1 of high–speed counter board					
DT9108 DT9109	Elapsed value of high–speed counter board					
DT9110 DT9111	Capture value of high–speed counter board					
DT9112 DT9113	Target value 0 of high–speed counter board	Channel 1		N/A	N/A	A
DT9114 DT9115	Target value 1 of high–speed counter board					
DT9116 DT9117	Elapsed value of high–speed counter board					
DT9118 DT9119	Capture value of high–speed counter board					
DT9120	High–speed counter board control register		This register is used to control the high–speed counter board by the **F0 (MV)** instruction.  Bit position 15 . . 12 11 . . 8 7 . . 4 3 . . 0 Data 0 0 1 0 CH0 Output mode for target 0 CH0 Internal reset control (1: reset) CH0 External reset control (1: disabled) CH0 “Target = Elapsed” output control (1: disabled) CH0 Target setting (1: set) Number system selection Set this bit to 1 (BIN number system) CH1 Output mode for target CH1 Internal reset control (1: reset) CH1 External reset control (1: disabled) CH1 “Target = Elapsed” output control (1: disabled) CH1 Target setting bit (1: set)	N/A	N/A	A

A: Available, N/A: Not available

<table>
<tr><th rowspan="3">Address</th><th rowspan="3">Name</th><th rowspan="3">Description</th><th colspan="3">Availability</th></tr>
<tr><th rowspan="2">FP1</th><th colspan="2">FP–M</th></tr>
<tr><th>C16</th><th>C20/
C32</th></tr>
<tr><td><b>DT9120</b></td><td><b>High–speed counter board control register</b></td><td>
<b>Output mode:</b>
The output goes on or off when the elapsed value becomes equal to the target. These bits specify the mode for output transition when the elapsed value beccomes equal to the target value. If the output mode is changed, set the target value again.
<table>
<tr><th>Bit position</th><th>Channel</th><th>Corresponding target value</th><th>Corresponding output</th></tr>
<tr><td><b>0</b></td><td rowspan="2">0</td><td>Target 0</td><td>OUT00</td></tr>
<tr><td><b>1</b></td><td>Target 1</td><td>OUT01</td></tr>
<tr><td><b>8</b></td><td rowspan="2">1</td><td>Target 0</td><td>OUT10</td></tr>
<tr><td><b>9</b></td><td>Target 1</td><td>OUT11</td></tr>
</table>
Bit data 0: off → on
1: on → off
<b>External reset control bit:</b>
These bits (bit position 3 and 11) are in the on state, the external reset inputs (RST.0/RST.1) are ignored as:

External reset control bit (bit positions 3 and 11)
on
off
External reset input (RST.0/RST.1)
on
off
Reset input ignored

By turning on the external reset enable inputs (RST.E0/RST.E1), you can enable the external reset inputs (RST.0/RST.1). The external reset inputs (RST.0/RST.1) effective are:
– external reset inputs while the external reset enable input is in the on states.
– the first external reset inputs after the external reset enable input turns off.

External reset control bit (bit positions 3 and 11)
on
off
External reset enable input (RST.E0/RST.E1)
on
off
External reset input (RST.0/RST.1)
on
off
Reset inputs become effective
</td><td>N/A</td><td>N/A</td><td>A</td></tr>
</table>

A: Available, N/A: Not available

Address	Name	Description	Availability		
			FP1	FP–M	
				C16	C20/ C32
DT9120	**High–speed counter board control register**	**Target setting:** To preset the target values for the high–speed counter board, first, transfer the set values to the special data registers for the target values. Then, turn the target setting bit from 0 to 1. A set value is revised at the moment the leading edge of this bit is detected. Therefore, if the bit is already set to 1, change the bit from 1 to 0 and then change it back to 1. **Number system selection:** This bit is prepared to select the number system used for the high–speed counter board. If you set this bit to 0, the data counts the number in the BCD code. However, the FP–M usually handles numbers in binary, so use of the binary number system is recommended.	N/A	N/A	A
DT9121	**High–speed counter board status register**	This register is stored the input and output conditions and error code of high–speed counter board.  Bit position 15 . . 12 11 . . 8 7 . . 4 3 . . 0 Data 0 0 0 CH0 Flag bit for “reset enable input (RST.E0 terminal)” [1: on (reset enabled)] CH0 Flag bit for “output disable input (O.INH0 terminal)” 0: off (output enabled) 1: on (output disabled) CH1 Flag bit for “reset enable input (RST.E1 terminal)” [1: on (reset enabled)] CH1 Flag bit for “output disable input (O.INH0 terminal)” 0: off (output enabled) 1: on (output disabled) CH0 This flag turns on when “Target 0 = elapsed value” CH0 This flag turns on when “Target 1 = elapsed value” CH1 This flag turns on when “Target 0 = elapsed value” CH1 This flag turns on when “Target 1 = elapsed value” Error code (see next page) Error flag (1: an error occurs)	N/A	N/A	A

A: Available, N/A: Not available

<table>
<tr><th rowspan="3">Address</th><th rowspan="3">Name</th><th rowspan="3">Description</th><th colspan="3">Availability</th></tr>
<tr><th rowspan="2">FP1</th><th colspan="2">FP–M</th></tr>
<tr><th>C16</th><th>C20/ C32</th></tr>
<tr><td><b>DT9121</b></td><td><b>High–speed counter board status register</b></td><td><b>Output disable input:</b>
This input disables external output even if the high–speed counter is set to the output enable mode by DT9120. While this input is turned on, the output of the high–speed counter board is not changed even if the elapsed value becomes equal to the target.

<b>Error codes:</b>
A BCD error is detected only when data for the high–speed couter board is set to BCD operation using <b>F0 (MV)</b> and bit position 7 of DT9120.

<table>
<tr><th colspan="4">Bit position</th><th rowspan="2">Description</th></tr>
<tr><th>11</th><th>10</th><th>9</th><th>8</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>BCD error</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>CH0 overflow/underflow</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>CH1 overflow/underflow</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>Watchdog timer error</td></tr>
</table>
</td><td>N/A</td><td>N/A</td><td>A</td></tr>
</table>

A: Available, N/A: Not available

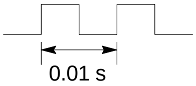
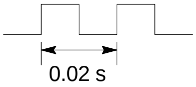
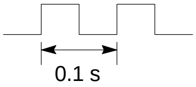
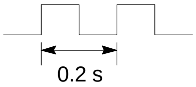
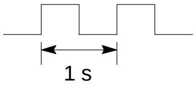
Appendix C

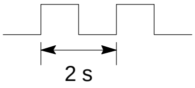
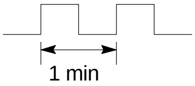
Special Internal Relays

C.1 Special Internal Relays

The special internal relays turn on and off under special conditions. The on and off states are not output externally. Writing is not possible with a programming tool or an instruction.

Address	Name	Description
R9000	**Self-diagnostic error flag** **(Available PLC: All types)**	Turns on when a self-diagnostic error occurs. The self-diagnostic error code is stored in: – FP–C/FP–M/FP0/FP1/FP3: DT9000 – FP2/FP2SH/FP10SH: DT90000
R9001	**Not used**	———
R9002	**MEWNET–TR master error flag** **(Available PLC: FP3, FP10SH)**	Turns on when a communication error occurs in the MEWNET–TR master unit or MEWNET–TR network. The slot, where the erroneous MEWNET–TR master unit is installed, can be checked using: – FP3: DT9002 and DT9003 – FP10SH: DT90002, DT90003
	I/O error flag **(Available PLC: FP2, FP2SH)**	Turns on when the error occurs in the I/O unit. The slot number of the unit where the error was occurred is stored in DT90002, DT90003.
R9003	**Intelligent unit error flag**	Turns on when an error occurs in an intelligent unit. The slot number, where the erroneous intelligent unit is installed is stored in: – FP–C/FP3: DT9006 and DT9007 – FP2/FP2SH/FP10SH: DT90006, DT90007
R9004	**I/O verification error flag**	Turns on when an I/O verification error occurs. The slot number of the I/O unit where the verification error was occurred is stored in: – FP–C/FP0/FP3: DT9010 and DT9011 – FP2/FP2SH/FP10SH: DT90010, DT90011
R9005	**Backup battery error flag (non-hold)** **(Available PLC: FP–C/ FP–M C20,C32/FP1 C24,C40,C56,C72/FP2/FP2 SH/FP3/FP10SH)**	Turns on for an instant when a backup battery error occurs.
R9006	**Backup battery error flag (hold)** **(Available PLC: FP–C/ FP–M C20,C32/FP1 C24,C40,C56,C72/FP2/FP2 SH/FP3/FP10SH)**	Turns on and keeps the on state when a backup battery error occurs. To reset R9006, – turn the power to off and then turn it on, – initialize, after removing the cause of error.
R9007	**Operation error flag (hold)**	Turns on and keeps the on state when an operation error occurs. The address where the error occurred is stored in: – FP–C/FP–M/FP0/FP1 CPU Ver.2.7 or later/FP3: DT9017 – FP2/FP2SH/FP10SH: DT90017 (indicates the first operation error which occurred).
R9008	**Operation error flag (non-hold)** **(Available PLC: FP–C/FP–M/FP1 CPU Ver.2.7 or later/FP2/FP2SH/FP10SH)**	Turns on for an instant when an operation error occurs. The address where the operation error occurred is stored in: – FP–C/FP–M/FP0/FP1 CPU Ver.2.7 or later/FP3: DT9018 – FP2/FP2SH/FP10SH: DT90018 The contents change each time a new error occurs.

Address	Name	Description
R9009	**Carry flag**	Turns on for an instant, – when an overflow or underflow occurs. – when “1” is set by one of the shift instructions.
R900A	**> flag**	Turns on for an instant when the compared results become larger in the **“F60 (CMP)/P60 (PCMP)**, **F61 (DCMP)/P61 (PDCMP)**, **F62 (WIN)/P62 (PWIN)** or **F63 (DWIN)/P63 (PDWIN)** comparison instructions.”
R900B	**= flag**	Turns on for an instant, – when the compared results are equal in the comparison instructions. – when the calculated results become 0 in the arithmetic instructions.
R900C	**< flag**	Turns on for an instant when the compared results become smaller in the **“F60 (CMP)/ P60 (PCMP)**, **F61 (DCMP)/P61 (PDCMP)**, **F62 (WIN)/P62 (PWIN)** or **F63 (DWIN)/P63 (PDWIN)** comparison instructions.”
R900D	**Auxiliary timer contact** **(Available PLC: FP–C/ FP–M C20,C32/FP0/FP1 C56,C72/FP2/FP2SHFP3/ FP10SH)**	Turns on when the set time elapses (set value reaches 0) in the timing operation of the **F137 (STMR)/F183 (DSTM)** auxiliary timer instruction. Available PLC for **F183(DSTM)** instruction: FP0/FP2/FP2SH/FP10SH CPU Ver.3.0. or later. The R900D turns off when the trigger for auxiliary timer instruction turns off.
R900E	**Tool port error flag** **(Available PLC: FP–M/ FP0/FP1/FP2SH/FP10SH)**	Turns on when communication error at tool port is occurred.
R900F	**Constant scan error flag**	Turns on when scan time exceeds the time specified in system register 34 during constant scan execution.
R9010	**Always on relay**	Always on.
R9011	**Always off relay**	Always off.
R9012	**Scan pulse relay**	Turns on and off alternately at each scan
R9013	**Initial on pulse relay**	Turns on only at the first scan in the operation. Turns off from the second scan and maintains the off state.
R9014	**Initial off pulse relay**	Turns off only at the first scan in the operation. Turns on from the second scan and maintains the on state.
R9015	**Step ladder initial on pulse relay**	Turns on for an instant only in the first scan of the process the moment the step ladder process is opened.
R9016	————————	Not used
R9017	————————	Not used
R9018	**0.01 s clock pulse relay**	Repeats on/off operations in 0.01 s cycles. 
R9019	**0.02 s clock pulse relay**	Repeats on/off operations in 0.02 s cycles. 
R901A	**0.1 s clock pulse relay**	Repeats on/off operations in 0.1 s cycles. 
R901B	**0.2 s clock pulse relay**	Repeats on/off operations in 0.2 s cycles. 
R901C	**1 s clock pulse relay**	Repeats on/off operations in 1 s cycles. 

Address	Name	Description
R901D	**2 s clock pulse relay**	Repeats on/off operations in 2 s cycles. 
R901E	**1 min clock pulse relay**	Repeats on/off operations in 1 min cycles. 
R901F	————————	Not used
R9020	**RUN mode flag**	Turns off while the mode selector is set to PROG. Turns on while the mode selector is set to RUN.
R9021	**Test RUN mode flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on while the initialize/test switch of the CPU is set to TEST and mode selector is set to RUN. (test run operation start) Turns off during the normal RUN mode.
R9022	**Break flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on while the **BRK** instruction is executing or the step run is executing.
R9023	**Break enable flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on while the **BRK** instruction is enabled in the test RUN mode.
R9024	**Output update enable flag in the test RUN mode** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on while the output update is enabled in the test RUN mode.
R9025	**Single instruction flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on while the single instruction execution is selected in the test RUN mode.
R9026	**Message flag** **(Available PLC: FP–M C20,C32/FP–C/FP0/FP1 C24,C40,C56,C72/FP2/FP2 SH/FP3/FP10SH)**	Turns on while the **F149 (MSG)/P149 (PMSG)** instruction is executed.
R9027	**Remote mode flag**	Turns on while the mode selector is set to REMOTE.
R9028	**Break clear flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on when the break operation is cleared.
R9029	**Forcing flag**	Turns on during forced on/off operation for I/O relay and timer/counter contacts.
R902A	**External interrupt enable flag** **(Available PLC: FP–M/ FP0/FP1 C24, C40, C56, C72/FP2SH/FP3/FP10SH)**	Turns on while the external interrupt trigger is enabled by the **ICTL** instruction.
	Interrupt flag **(Available PLC: FP2)**	Turns on while the periodical interrupt is executed by the **ICTL** instruction.
R902B	**Interrupt error flag** **FP–M/FP0/FP1 C24, C40, C56, C72/FP2/FP2SH/FP3/ FP10SH**	Turns on when an interrupt error occurs.
R902C	**Sampling point flag**	Turns off during instructed sampling. Turns on while sampling is triggered by the periodical interrupt.

Address	Name	Description
R902D	**Sampling trace end flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on when the sampling trace ends.
R902E	**Sampling trigger flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on when the sampling trace trigger of the **F156 (STRG)/P156 (PSTGR)** instruction is turned on.
R902F	**Sampling enable flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Turns on when the starting point of sampling is specified.
R9030	**F145 (SEND)/P145 (PSEND) and F146 (RECV)/P146 (PRECV) instruction executing flag**	Monitors if CPU is in the **F145 (SEND)/P145 (PSEND)** and **F146 (RECV)/P146 (PRECV)** instructions executable condition as follows: – off: None of the above mentioned instructions can be executed. – on: One of the above mentioned instructions can be executed.
R9031	**F145 (SEND)/P145 (PSEND) and F146 (RECV)/P146 (PRECV) instruction end flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Monitors if an abnormality has been detected during the execution of the **F145 (SEND)/ P145 (PSEND)** and **F146 (RECV)/P146 (PRECV)** instructions as follows: – off: No abnormality detected. – on: An abnormality detected. (communication error) The error code is stored in: – FP–C/FP3: DT9039 – FP2/FP10SH: DT90039
R9032	**COM port mode flag** **(Available PLC:FP–M C20C,C32C/FP0/FP1 C24C,C40C,C56C,C72C/ FP2/FP2SH/FP10SH)**	Monitors the mode of the COM port as: – on: Serial data communication mode – off: Computer link mode
R9033	**F147 (PR) instruction flag** **(Available PLC: FP–M C20,C32/FP–C/FP0/FP1 C24,C40,C56,C72/FP2/FP2 SH/FP3/FP10SH)**	Turns on while a **F147 (PR)** instruction is executed. Turns off when a **F147 (PR)** instruction is not executed.
R9034	**Editing in RUN mode flag** **(Available PLC: FP–C/FP0 CPU Ver.2.0 or later/ FP2/FP2SH/FP3/FP10SH)**	Turns on while editing a program in the RUN mode.
R9035	**F152 (RMRD)/P152 (PRMRD) and F153 (RMWT)/P153 (PRMWT) instruction execution flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Monitors if FP3/FP10SH is in the **F152 (RMRD)/P152 (PRMRD)** and **F153 (RMWT)/P153 (PRMWT)** instructions executable condition as follows: – off: None of the above mentioned instructions can be executed. – on: One of the above mentioned instructions can be executed.
	S–LINK I/0 communication error flag (Available PLC: FP0)	Tuns on when the S–LINK error (ERR1, 3 or 4) occurs using S–LINK system.

Address	Name	Description
R9036	**F152 (RMRD)/P152 (PRMRD) and F153 (RMWT)/P153 (PRMWT) instruction end flag** **(Available PLC: FP–C/ FP2/FP2SH/FP3/FP10SH)**	Monitors if an abnormality has been detected during the execution of the **F152 (RMRD)/P152 (PRMRD)** and **F153 (RMWT)/P153 (PRMWT)** instructions as follows: – off: No abnormality detected. – on: An abnormality detected. (access error) The error code is stored in: – FP–C/FP3: DT9036 – FP2/FP2SH/FP10SH: DT90036
	I/0 link error flag **(Available PLC: FP–M C20,C32/FP1)**	Turns on when the error occurs using the I/0 link function.
R9037	**COM (RS232C) port communication error flag** **(Available PLC: FP–M C20C,C32C/FP0/FP1 C24C,C40C,C56C,C72C/ FP0/FP2/FP2SH/FP10SH)**	Turns on when the serial data communication error occurs using COM port. Turns off when data is being sent by the **F144 (TRNS)** instruction.
R9038	**COM (RS232C) port receive flag** **(Available PLC: FP–M C20C,C32C/FP0/FP1 C24C,C40C,C56C,C72C/ FP2/FP2SH/FP10SH)**	Tuns on when the end code is received during the serial data communicating.
R9039	**COM (RS232C) port send flag** **(Available PLC: FP–M C20C,C32C/FP0/FP1 C24C,C40C,C56C,C72C/ FP2/FP2SH/FP10SH)**	Tuns on while data is not send during the serial data communicating. Tuns off while data is being sent during the serial data communicating.
R903A	**High–speed counter control flag (ch 0) (Available PLC: FP–M C20,C32/FP0/FP1)**	Turns on while the high–speed counter instructions “**F166 (HC1S)** to **F170 (PMW)**” is executed.
R903B	**Cam control flag (Available PLC: FP–M/FP1)**	Tuns on while the cam control instruction “**F165 (CAMO)**” is executed.
	High–speed counter control flag (for ch1)	Turns on while the high–speed counter instruction “**F166 (HC1S)** to **F170 (PWM)**” is executed.
R903C	**High–speed counter control flag (for ch2)**	Turns on while the high–speed counter instruction “**F166 (HC1S)** to **F170 (PWM)**” is executed.
R903D	**High–speed counter control flag (for ch3)**	Turns on while the high–speed counter instruction “**F166 (HC1S)** to **F170 (PWM)**” is executed.
R903E	———	Not used
R903F	———	Not used
R9040	**Error alarm (D to 2047)**	Turns on while the error alarm relay (E0 to E2047) acts. Tuns off when the all error alarm relay turns off.

Appendix D

Relays, Memory Areas and Constants

D.1 Relays

Item		Function	Numbering		
			FP–M		
			C16T	C20R/ C20T/ C32T	C20RC/ C20TC/ C32TC
Relay	**External input relay (X)**	Turn on or off based on external input.	208 points (X0 to X12F)		
	External output relay (Y)	Externally outputs on or off state.	208 points (Y0 to Y12F)		
	Internal relay (R) (* Note 1)	Relay which turns on or off only within program.	256 points (R0 to R15F)	1,008 points (R0 to R62F)	
	Link relay (L)	This relay is a shared relay used for MEWNET link system.	————		
	Timer (T) (* Notes 1 and 2)	If a **TM** instruction has timed out, the contact with the same number turns on.	128 points (T0 to T99/ C100 to C127) (*Note 2)	144 points (T0 to T99/C100 to C143) (*Note 2)	
	Counter (C) (* Notes 1 and 2)	If a **CT** instruction has counted up, the contact with the same number turns on.			
	Pulse relay (P)	This relay is used to turn on only for one scan duration programmed with the **OT**↑ and **OT**↓ instructions.	————		
	Error alarm relay (E)	If turned on while the unit is running, this relay stores the history in a dedicated buffer.Program this relay so that it is turned on at the time of abnormality.	————		
	Special internal relay (R)	Relay which turns on or off based on specific conditions and is used as a flag.	64 points (R9000 to R903F)		

1) There are two unit types, the hold type that saves the conditions that exist just before turning the power off or changing from the RUN mode to PROG. mode, and the non–hold type that resets them. These areas can be specified as hold type or non–hold type by setting system register.
For the FP0 T32C/FP–M/FP1, the selection of hold type and non–hold type can be changed by the setting of system register. For the FP0 C10/C14/C16/C32 series, that area is fixed and allotted tha numbers as shown below.
Hold type and non–hold type areas are listed in the table on the bottom of the next page.

Numbering					
FP0			**FP1**		
C10/C14/C16	**C32**	**T32C**	**C14/C16**	**C24/C40**	**C56/C72**
208 points (X0 to X12F)			208 points (X0 to X12F)		
208 points (Y0 to Y12F)			208 points (X0 to X12F)		
1,008 points (R0 to R62F)			256 points (R0 to R15F)	1,008 points (R0 to R62F)	
———			———		
144 points (T0 to T99/C100 to C143) (*Note 1)			128 points (T0 to T99/ C100 to C127) (*Note 2)	144 points (T0 to T99/C100 to C143) (*Note 2)	
———			———		
———			———		
64 points (R9000 to R903F)			64 points (R9000 to R903F)		

Timer		Non–hold type: All points	
Counter	**Non-hold type**	From the set value to C139	From the set value to C127
	Hold type	4 points (elapsed values) (C140 to C143)	16 points (elapsed values) C128 to C143
Internal relay	**Non-hold type**	976 points (R0 to R60F) 61 words (WR0 to WR60)	880 points (R0 to R54F) 55 words (WR0 to WR54)
	Hold type	32 points (R610 to R62F) 2 words (WR61 to WR62)	128 points (R550 to R62F) 8 words (WR55 to WR62)
Data register	**Non-hold type**	1652 words (DT0 to DT1651)	6112 words (DT0 to DT6111)
	Hold type	8 words (DT1652 to DT1659)	32 words (DT6112 to DT6143)

2) The points for the timer and counter can be changed by the setting of system register 5. The number given in the table are the numbers when system register 5 is at its default setting.

D.2 Memory Areas

☞ **The data in the following table is based on 16–bit memory areas. You may also access these memory areas in 32–bit increments. The letter “D” in front of the Matsushita address indicates this.**

Example **If you access the Matsushita address DDT0 (IEC–Adresse %MD5.0), the program accesses the addresses DT0 + DT1, i.e. the data is processed in double–word units.**

Item		**Function**	**Numbering**		
			FP–M		
			C16T	**C20R/ C20T/ C32T**	**C20RC/ C20TC/ C32TC**
Memory area	**External input relay (WX)**	Code for specifying 16 external input points as one word (16 bits) of data.	13 words (WX0 to WX12)		
	External output relay (WY)	Code for specifying 16 external output points as one word (16 bits) of data.	13 words (WY0 to WY12)		
	Internal relay (WR)	Code for specifying 16 internal relay points as one word (16 bits) of data.	16 words (WR0 to WR15)	63 words (WR0 to WR62)	
	Link relay (WL)	Code for specifying 16 link relay points as one word (16 bits) of data.	———		
	Data register (DT) (* Note 1)	Data memory used in program. Data is handled in 16-bit units (one word).	256 words (DT0 to DT255)	1,660 wprds (DT0 to DT1659)	6,144 words (DT0 to DT6143)
	Link data register (LD) (* Note 1)	This is a shared data memory which is used within the MEWNET link system. Data is handled in 16-bit units (one word).	———		
	Timer/Counter set value area (SV) (* Note 1)	Data memory for storing a target value of a timer and an initial value of a counter. Stores by timer/counter number.	128 words (SV0 to SV127)	144 words (SV0 to SV143)	
	Timer/Counter elapsed value area (EV) (* Note 1)	Data memory for storing the elapsed value during operation of a timer/counter. Stores by timer/counter number.	128 words (EV0 to EV127)	144 words (EV0 to EV143)	
	File register (FL) (* Note 1)	Data memory used in program. Data is handled in 16-bit units (one word).	———		
	Special data register (DT)	Data memory for storing specific data. Various settings and error codes are stored.	70 words (DT9000 to DT9069)	112 words (DT9000 to DT9069) (DT9080 to DT9121)	
	Index register (I)	Register can be used as an address of memory area and constants modifier.	2 words (IX, IY)		

Numbering					
FP0			**FP1**		
C10/C14/C16	**C32**	**T32C**	**C14/C16**	**C24/C40**	**C56/C72**
13 words (WX0 to WX12)			13 words (WX0 to WX12)		
13 words (WY0 to WY12)			13 words (WY0 to WY12)		
63 words (WR0 to WR62)			16 words (WR0 to WR15)	63 words (WR0 to WR62)	
—			—		
1,660 words (DT0 to DT1659)	6,144 words (DT0 to DT6143)	16,384 words (DT0 to DT16383)	256 words (DT0 to DT255)	1,660 words (DT0 to DT1659)	6,144 words(DT0 to DT6143)
—			—		
144 words (SV0 to SV143)			128 words (SV0 to SV127)	144 words (SV0 to SV143)	
144 words (EV0 to EV143)			128 words (EV0 to EV127)	144 words (EV0 to EV143)	
—			—		
112 words (DT9000 to DT9111)		112 words (DT90000 to DT90111)	70 words (DT 9000 to DT 9069)		
2 words (IX, IY)			2 words (IX, IY)		

D.3 Constants

Item		Numbering		
		FP–M		
		C16T	C20R/ C20T/ C32T	C20RC/ C20TC/ C32TC
Constant	Decimal constants (K) (integer type)	K–32768 to K32767 (for 16-bit operation)		
		K–2147483648 to K2147483647 (for 32-bit operation)		
	Hexadecimal (H) constants	H0 to HFFFF (for 16-bit operation)		
		H0 to HFFFFFFFF (for 32-bit operation)		
	Decimal constants (f) (monorefined real number)	—		

Numbering					
FP0			FP1		
C10/C14/C16	C32	T32C	C14/C16	C24/C40	C56/C72
K–32768 to K32767 (for 16-bit operation)			K–32768 to K32767 (for 16-bit operation)		
K–2147483648 to K2147483647 (for 32-bit operation)			K–2147483648 to K2147483647 (for 32-bit operation)		
H0 to HFFFF (for 16-bit operation)			H0 to HFFFF (for 16-bit operation)		
H0 to HFFFFFFFF (for 32-bit operation)			H0 to HFFFFFFFF (for 32-bit operation)		
—			—		

Appendix E

System Registers

E.1 System Registers for FP0

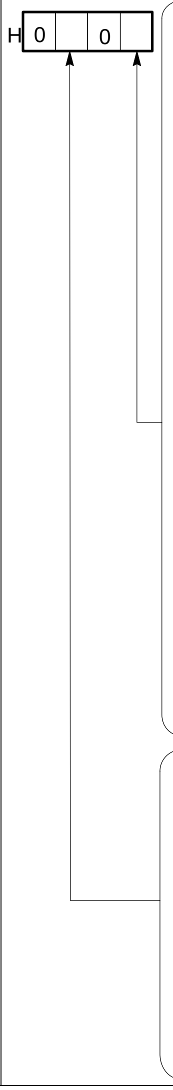
C10, C14, C16, C32 and T32C in the table respectively indicate 10-point, 14-point, 16-point and 32-point type FP0 control units.

Item	Address	Name	Default value	Description
Allocation of user memory	**0**	**Sequence program area capacity**	—	The set values are fixed and cannot be changed. The stored values vary depending on the type. K3: 3K words (FP0 C10, C14, C16) K5: 5K words (FP0 C32) K10: 10K words (FP0 T32C)
	1 to 3	**Not used**	—	—
Hold/ Non–hold	**5**	**Timer and counter division (setting of starting counter number)**	K100	K0 to K144
	6 to 8	**Not used (Available type: C10, C14, C16, C32)**	—	With the FP0 C10/C14/C16/C32, values set with the programming tool become invalid.
	6	**Hold type area starting number setting for timer and counter**	K100	K0 to K144 Set the system registers 5 and 6 to the same value.
	7	**Hold type area starting number setting for internal relays (in word units)**	K10	K0 to K63
	8	**Hold type area starting number setting for data registers**	K0	K0 to K16384
	9 to 13	**Not used**	—	—
	14	**Not used (Available type: C10, C14, C16, C32)**	—	With the FP0 C10/C14/C16/C32, values set with the programming tool become invalid.
		Hold or non–hold setting for step ladder process	K1	K0: Hold K1: Non–hold
	15	**Not used**	—	—
Action on error	**20**	**Disable or enable setting for duplicated output**	K0	K0: Disable (will be syntax error) K1: Enable (will not be syntax error)
	21, 22	**Not used**	—	—
	23	**Operation setting when an I/O verification error occurs**	K0	K0: Stop K1: Continuation
	24, 25	**Not used**	—	—
	26	**Operation setting when an operation error occurs**	K0	K0: Stop K1: Continuation
	27	**Operation settings when communication error occurs in the remote I/O (S–LINK) system**	K1	K0: Stop K1: Continuation
	28, 29	**Not used**	—	—
	4	**Not used**	—	With the FP0, values set with the programming tool become invalid.

Item	Address	Name	Default value	Description
Time setting	**30**	**Not used**	——	——
	31	**Wait time setting for multi-frame communication**	K2600 (6500ms)	K4 to K32760: 10ms to 81900ms Used of default setting (K2600/ 6500ms) is recommended. set value X 2.5ms = Wait time setting for multi–frame communication (ms) In programming tool software, enter the time (a number divisible by 2.5). In FP Programmer II, enter the set value (equal to the time divided by 2.5).
	32, 33	**Not used**	——	With the FP0, values set with the programming tool become invalid.
	34	**Constant value settings for scan time**	K0	K1 to K64 (2.5ms to 160ms): Scans once each specified time interval. K0: Normal scan set value X 2.5ms = Constant value setting for scan time (ms) In programming tool software, enter the time (a number divisible by 2.5). In FP Programmer II, enter the set value (equal to the time divided by 2.5).

Item	Address	Name		Default value	Description	
Input setting	**400**	**High-speed counter mode settings (X0 to X2)**	**Setting by programming tool software**	H0	CH0	0: Do not set input X0 as high-speed counter. 1: 2-phase input (X0, X1) 2: 2-phase input (X0, X1), Reset input (X2) 3: Incremental input (X0) 4: Incremental input (X0), Reset input (X2) 5: Decremental input (X0) 6: Decremental input (X0), Reset input (X2) 7: Individual input (X0, X1) 8: Individual input (X0, X1), Reset input (X2) 9: Direction decision (X0, X1) 10:Direction decision (X0, X1), Reset input (X2)
					CH1	0: Do not set input X1 as high-speed counter. 3: Incremental input (X1) 4: Incremental input (X1), Reset input (X2) 5: Decremental input (X1) 6: Decremental input (X1), Reset input (X2)

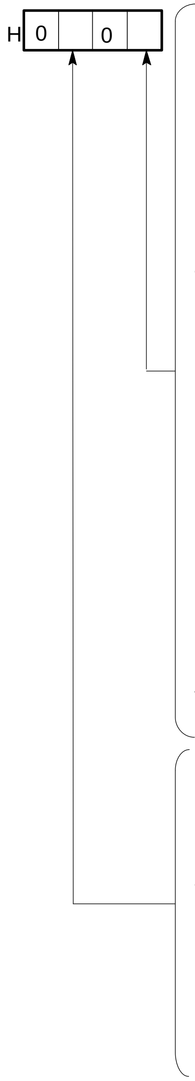
- **If the operation mode is set to 2–phase, individual, or direction differentiation, the setting for CH1 is invalid.**
- **If reset input settings overlap, the setting of CH1 takes precedence.**
- **If system register 400 to 403 have been set simultaneously for the same input relay, the following precedence order is effective: [High–speed counter] ➧ [Pulse catch] ➧ [Interrupt input].**

Item	Address	Name		Default value	Description	
Input setting	**400**	**High-speed counter mode settings (X0 to X2)**	**Setting by FP programmer II**	H0	CH0/ CH1	 0: Do not use high-speed counter. 1: 2-phase input (X0, X1) 2: 2-phase input (X0, X1), Reset input (X2) 3: Incremental input (X0) 4: Incremental input (X0), Reset input (X2) 5: Decremental input (X0) 6: Decremental input (X0), Reset input (X2) 7: Individual input (X0, X1) 8: Individual input (X0, X1), Reset input (X2) 9: Direction decision (X0, X1) A: Direction decision (X0, X1), Reset input (X2) 0: Do not use high-speed counter. 3: Incremental input (X1) 4: Incremental input (X1), Reset input (X2) 5: Decremental input (X1) 6: Decremental input (X1), Reset input (X2)

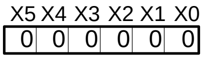
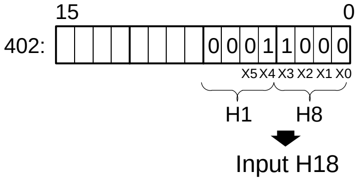
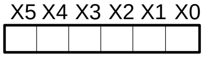
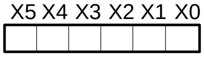
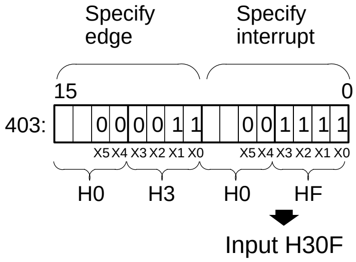
- **If the operation mode is set to 2–phase, individual, or direction differentiation, the setting for CH1 is invalid.**
- **If reset input settings overlap, the setting of CH1 takes precedence.**
- **If system register 400 to 403 have been set simultaneously for the same input relay, the following precedence order is effective: [High–speed counter] ➧ [Pulse catch] ➧ [Interrupt input].**

Item	Address	Name		Default value	Description	
Input setting	**401**	**High-speed counter mode settings (X3 to X5)**	**Setting by programming tool software**	H0	CH2	0: Do not set input X3 as high-speed counter. 1: 2-phase input (X3, X4) 2: 2-phase input (X3, X4), Reset input (X5) 3: Incremental input (X3) 4: Incremental input (X3), Reset input (X5) 5: Decremental input (X3) 6: Decremental input (X3), Reset input (X5) 7: Individual input (X3, X4) 8: Individual input (X3, X4), Reset input (X5) 9: Direction decision (X3, X4) 10: Direction decision (X3, X4), Reset input (X5)
					CH3	0: Do not set input X4 as high-speed counter. 3: Incremental input (X4) 4: Incremental input (X4), Reset input (X5) 5: Decremental input (X4) 6: Decremental input (X4), Reset input (X5)

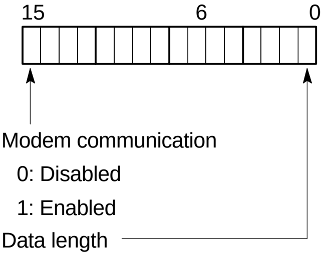
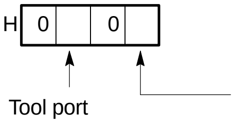
- **If the operation mode is set to 2–phase, individual, or direction differentiation, the setting for CH3 is invalid.**
- **If reset input settings overlap, the setting of CH3 takes precedence.**
- **If system register 400 to 403 have been set simultaneously for the same input relay, the following precedence order is effective: [High–speed counter] ➧ [Pulse catch] ➧ [Interrupt input].**

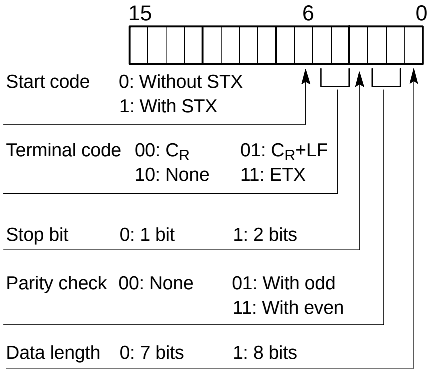
Item	Address	Name		Default value	Description	
Input setting	**401**	**High-speed counter mode settings (X3 to X5)**	**Setting by FP programmer II**	H0	CH2/ CH3	 0: Do not use high-speed counter. 1: 2-phase input (X3, X4) 2: 2-phase input (X3, X4), Reset input (X5) 3: Incremental input (X3) 4: Incremental input (X3), Reset input (X5) 5: Decremental input (X3) 6: Decremental input (X3), Reset input (X5) 7: Individual input (X3, X4) 8: Individual input (X3, X4), Reset input (X5) 9: Direction decision (X3, X4) A: Direction decision (X3, X4), Reset input (X5) 0: Do not use high-speed counter. 3: Incremental input (X4) 4: Incremental input (X4), Reset input (X5) 5: Decremental input (X4) 6: Decremental input (X4), Reset input (X5)

- **If the operation mode is set to 2–phase, individual, or direction differentiation, the setting for CH3 is invalid.**
- **If reset input settings overlap, the setting of CH3 takes precedence.**
- **If system register 400 to 403 have been set simultaneously for the same input relay, the following precedence order is effective: [High–speed counter] ➧ [Pulse catch] ➧ [Interrupt input].**

Item	Address	Name	Default value	Description
Input setting	**402**	**Pulse catch input function settings**	H0	 0: Standard input 1: Pulse catch input In FP Programmer II, enter the above settings in hexadecimal. Example: When X3 and X4 are set to pulse catch input  15 0 402: 0 0 0 1 1 0 0 0 X5 X4 X3 X2 X1 X0 H1 H8 Input H18 In the case of FP0, settings X6 and X7 are invalid.
	403	**Interrupt input settings**	H0	Using programming tool software  Specify the input contacts used as interrupt inputs in the upper byte. (0: Standard input/1: Interrupt input)  Specify the effective interrupt edge in the lower byte. (When 0: on/When 1: off) Using FP programmer II Example: When setting inputs X0, X1, X2, and X3 as interrupts, and X0 and X1 are set as interrupt inputs when going from on to off.  Specify edge Specify interrupt 15 0 403: 0 0 0 0 1 1 0 0 1 1 1 1 X5 X4 X3 X2 X1 X0 X5 X4 X3 X2 X1 X0 H0 H3 H0 HF Input H30F
	404 to 407	**Not used**	——	With the FP0, values set with the programming tool become invalid.

If system register 400 to 403 are set simultaneously for the same input relay, the following precedence order is effective: [High–speed counter] → [Pulse catch] → [Interrupt input].
When the high–speed counter is being used in the incremental input mode, even if input X0 is specified as an interrupt input and as pulse catch input, those settings are invalid, and input X0 functions as counter input for the high–speed counter.
No. 400: H1 ← This setting will be valid.
No. 402: H1
No. 403: H1

Item	Address	Name		Default value	Description
Tool port setting	**410**	**Unit number setting for tool port (when connecting C–NET)**		K1	K1 to K32 (Unit No. 1 to 32)
	411	**Communication format setting for tool port** **Default setting Item** **• Modem: Disabled** **• Data length: 8 bits**		H0	Using programming tool software Select items from the menu. Using FP programmer II Specify the setting contents using H constants.  15 6 0 Modem communication 0: Disabled 1: Enabled Data length (character bits) 0: 8 bits 1: 7 bits When connecting a modem, set the unit number to 1 with system resister 410.
	414	**Baud rate setting for tool port**		H0	0: 9600bps 1: 19200bps
Tool port/ RS232C port setting	**414**	**Baud rate setting for tool port and RS232C port**	**Setting by FP programmer II**	H1	 H 0 0 Tool port H0: 9600bps H1: 19200bps (If anything other than H0 or H1 is set for the baud rate of tool port, the baud rate will be 9600bps.) RS232C port H0: 19200bps H1: 9600bps H2: 4800bps H3: 2400bps H4: 1200bps H5: 600bps H6: 300bps Example: If 19,200bps is set for both the tool port and RS232C port ➧ H100 should be written.

Item	Address	Name		Default value	Description
RS232C port setting	**412**	**Communication method setting for RS232C port**		K0	Using programming tool software Select items from the menu. Using FP programmer II K0: RS232C port is not used. K1: Computer link communication mode (when connecting C–NET) K2: Serial data communication mode (general port)
	413	**Communication format setting for RS232C port** **Setting item/Default setting value** **– Start code: Without STX** **– Terminal code: C_R** **– Stop bit: 1 bit** **– Parity check: With odd** **– Data length: 8 bits**		H3	Using programming tool software Select items from the menu. Using FP programmer II Specify the setting contents using H constants.  15 6 0 Start code 0: Without STX 1: With STX Terminal code 00: C_R 01: C_R+LF 10: None 11: ETX Stop bit 0: 1 bit 1: 2 bits Parity check 00: None 01: With odd 11: With even Data length 0: 7 bits 1: 8 bits
	414	**Baud rate setting for RS232C port**		H1	0: 19200bps 1: 9600bps 2: 4800bps 3: 2400bps 4: 1200bps 5: 600bps 6: 300bps
	415	**Unit number setting for RS232C port (when connecting C–NET)**		K1	K1 to K32 (unit No. 1 to 32)
	416	**Modem compatibility setting for RS232C port**		H0	Using programming tool software Select items from the menu. Using FP programmer II H0: Modem disabled H8000: Modem enabled
	417	**Starting address setting for received buffer**		K0	C10C/C14C/C16C type: K0 to K1660 C32C type: K0 to K6144 T32C type: K0 to K16383
	418	**Capacity setting for received buffer**	**C10C/ C14C/ C16C type**	K1660	K0 to K1660
			C32C type	K6144	K0 to K6144
			T32C type	K16384	K0 to K16384

E.2 System Registers for FP–M/FP1

Item	Address	Name	Default value	Description
Allocation of user memory	**0**	**Sequence program area capacity**	—	The set values are fixed and cannot be changed. The stored values vary depending on the type. K1: FP1 C14/C16, FP–M C16T K3: FP1 C24/C40, FP–M 2.7K K5: FP1 C56/C72, FP–M 5K
	1 to 3	**Not used**	—	—
Action on error	**4**	**(Available for CPU version 2.7 or later)**	K0	K0: Enabled (R9000, R9005 and R9006 turn on, ERROR LED lights.) K1: Disabled
Hold/ Non–hold	**5**	**Timer and counter division (setting of starting counter number)**	K100	K0 to K144 (FP1 C14/C16, FP–M C16T: K0 to K128)
	6	**Hold type area starting number setting for timer and counter**	K100	K0 to K144 (FP1 C14/C16, FP–M C16T: K0 to 128) Set the system regis-ters 5 and 6 to the same value.
	7	**Hold type area starting number setting for internal relays (in word units)**	K10	(FP1 C14/C16, FP–M C16T: K0 to K16)
	8	**Hold type area starting number setting for data registers**	K0	K0 to K256 (FP1 C14/C16, FP–M C16T) K0 to K1660 (FP1 C24/C40, FP–M 2.7K) K0 to K6144 (FP1 C56/C72, FP–M 5K)
	9 to 13	**Not used**	—	—
	14	**Hold or non–hold setting for step ladder process**	K1	K0: Hold K1: Non–hold
	15 to 19	**Not used**	—	—
Action on error	**20**	**Disable or enable setting for duplicated output**	K0	K0: Disable (will be syntax error) K1: Enable (will not be syntax error)
	21 to 25	**Not used**	—	—
	26	**Operation setting when an operation error occurs**	K1	K0: Stop K1: Continuation
	27 to 29	**Not used**	—	—

Item	Address	Name	Default value	Description
Time setting	**30**	**Not used**	—	—
	31	**Wait time setting for multi-frame communication**	K2600 (6500ms)	K4 to K32760: 10ms to 81900ms Used of default setting (K2600/ 6500ms) is recommended. set value X 2.5 = Wait time setting for multi–frame communication (ms) In programming tool software, enter the time (a number divisible by 2.5). In FP Programmer II, enter the set value (equal to the time divided by 2.5).
	32, 33	**Not used**	—	—
	34	**Constant value settings for scan time**	K0	K1 to K64 (2.5ms to 160ms): Scans once each specified time interval. K0: Normal scan set value X 2.5 = Constant value setting for scan time (ms) In programming tool software, enter the time (a number divisible by 2.5). In FP Programmer II, enter the set value (equal to the time divided by 2.5).

<table>
<tr><th>Item</th><th>Address</th><th>Name</th><th>Default value</th><th>Description</th></tr>
<tr><td rowspan="2"><b>Input setting</b></td><td rowspan="2"><b>400</b></td><td><b>High–speed counter mode settings</b></td><td rowspan="2">H0</td><td rowspan="2">H □ 0 □
High–speed counter mode
<table>
<tr><th rowspan="2">Set value</th><th colspan="3">Input contact of FP–Ms and FP1s</th></tr>
<tr><th>X0</th><th>X1</th><th>X2</th></tr>
<tr><td><b>H0</b></td><td colspan="3">High–speed counter function not used.</td></tr>
<tr><td><b>H1</b></td><td colspan="2">2–phase input</td><td>———</td></tr>
<tr><td><b>H2</b></td><td colspan="2">2–phase input</td><td>Reset input</td></tr>
<tr><td><b>H3</b></td><td>Up input</td><td colspan="2">———</td></tr>
<tr><td><b>H4</b></td><td>Up input</td><td>———</td><td>Reset input</td></tr>
<tr><td><b>H5</b></td><td>———</td><td>Down input</td><td>———</td></tr>
<tr><td><b>H6</b></td><td>———</td><td>Down input</td><td>Reset input</td></tr>
<tr><td><b>H7</b></td><td colspan="2">Up/Down input
(X0: Up input, X1: Down input)</td><td>———</td></tr>
<tr><td><b>H8</b></td><td colspan="2">Up/Down input
(X0: Up input, X1: Down input)</td><td>Reset input</td></tr>
</table>
Pulse output internal connection
H0: Internally not connected
H1: Internally connected</td></tr>
<tr><td><b>Internal connection setting for pulse output
(FP1 C56/C72, FP–M only)</b></td></tr>
</table>

When system registers 400, 402, 403, and 404 are set at the same time, their priorities are:

- **– 1st 400 (high–speed counter mode settings)**
- **– 2nd 402 (pulse catch input function settings)**
- **– 3rd 403 (interrupt trigger settings)**
- **– last 404 (input time filtering settings)**

<table>
<tr><th>Item</th><th>Address</th><th>Name</th><th>Default value</th><th>Description</th></tr>
<tr><td><b>Input setting</b></td><td><b>401</b></td><td><b>Not used</b></td><td>——</td><td>——————</td></tr>
<tr><td></td><td><b>402</b></td><td><b>Pulse catch input function settings</b></td><td>H0</td><td>
FP1 C14/C16: X0 to X3
FP1 C24/C40/C56/C72: 0 to X7
FP–M C20/C32T: X0 to X7

System register 402
<table>
<tr><th>Bit position</th><th>15 · · 12</th><th>11 · · 8</th><th>7 · · 4</th><th>3 · · 0</th></tr>
<tr><td><b>Corresponding input</b></td><td>——</td><td>——</td><td>X7 X6 X5 X4</td><td>X3 X2 X1 X0</td></tr>
</table>
0: Standard input
1: Pulse catch input

In the FP programmer II, enter the above setting in hexadecimal.

Example: If the pulse catch function is used for inputs X3 and X4, input H18.

System register 402
<table>
<tr><th>Bit position</th><th>15 · · 12</th><th>11 · · 8</th><th>7 · · 4</th><th>3 · · 0</th></tr>
<tr><td><b>Corresponding input</b></td><td>——</td><td>——</td><td>X7 X6 X5 X4</td><td>X3 X2 X1 X0</td></tr>
<tr><td><b>Data input</b></td><td>0 0 0 0</td><td>0 0 0 0</td><td>0 0 0 1</td><td>1 0 0 0</td></tr>
<tr><td></td><td></td><td></td><td>H1</td><td>H8</td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td></td><td></td><td>
FP1 C14/C16 and FP–M C16T: X0 to X3
<table>
<tr><th>X3</th><th>X2</th><th>X1</th><th>X0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
0: Standard input
1: Pulse catch input

In the FP programmer II, enter the above setting in hexadecimal.

Example: If the pulse catch input function is used for input X3, input H8.
<table>
<tr><th></th><th>15</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>0</th></tr>
<tr><td>402:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>X3</td><td>X2</td><td>X1</td><td>X0</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>H8</td><td></td><td></td><td></td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Item</th><th>Address</th><th>Name</th><th>Default value</th><th>Description</th></tr>
<tr><td rowspan="2"><b>Input setting</b></td><td rowspan="2"><b>403</b></td><td rowspan="2"><b>Interrupt input settings</b></td><td rowspan="2">H0</td><td>
<table>
<tr><th>Bit position</th><th>15 · · 12</th><th>11 · · 8</th><th>7 · · 4</th><th>3 · · 0</th></tr>
<tr><td><b>Corresponding input</b></td><td>———</td><td>———</td><td>X7 X6 X5 X4</td><td>X3 X2 X1 X0</td></tr>
</table>
0: Standard input mode
1: Interrupt input mode

In the FP programmer II, enter the above setting in hexadecimal.

Example: If the interrupt input function is used for inputs X5 to X7, input HE0.

System register 403
<table>
<tr><th>Bit position</th><th>15 · · 12</th><th>11 · · 8</th><th>7 · · 4</th><th>3 · · 0</th></tr>
<tr><td><b>Corresponding input</b></td><td>———</td><td>———</td><td>X7 X6 X5 X4</td><td>X3 X2 X1 X0</td></tr>
<tr><td><b>Data input</b></td><td>0 0 0 0</td><td>0 0 0 0</td><td>1 1 1 0</td><td>0 0 0 0</td></tr>
<tr><td></td><td></td><td></td><td>HE</td><td>H0</td></tr>
</table>
• FP1 C14/C16 series: Not available
</td></tr>
<tr><td>
FP–M C16T: 2 inputs X4 and X5
By setting the interrupt input, the input program is executed when the interrupt input turns on during RUN (the <b>ICTL</b> instruction cannot be used).

<table>
<tr><th></th><th></th><th>X5</th><th>X4</th><th></th><th></th><th></th><th></th></tr>
<tr><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td></tr>
</table>
0: Standard input
1: Interrupt input

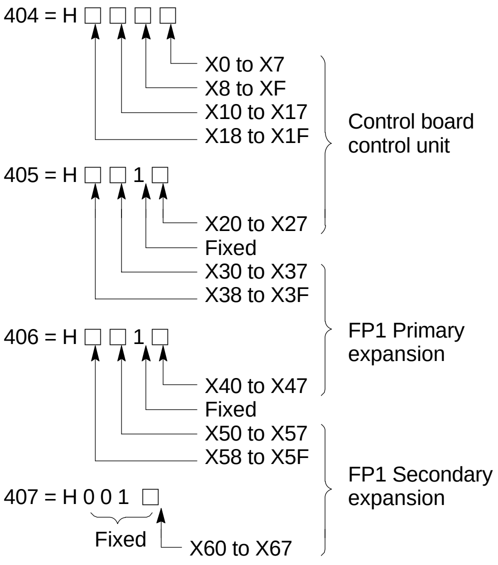
In the FP programmer II, enter the above setting in hexadecimal.

Example: If the interrupt input is used for input X5, input H20.

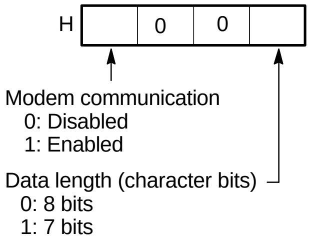
<table>
<tr><th></th><th>15</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>0</th></tr>
<tr><td>403:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>X5</td><td>X4</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">H2</td><td colspan="4">H0</td></tr>
</table>
</td></tr>
</table>

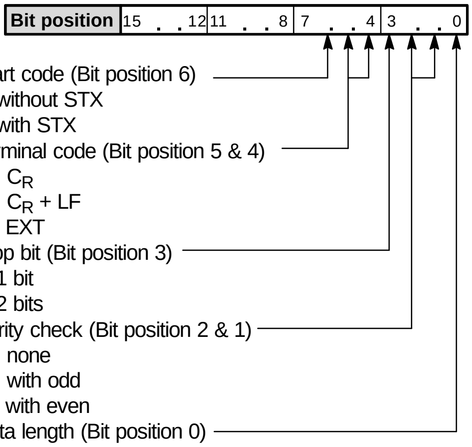
<table>
<tr><th>Item</th><th>Address</th><th>Name</th><th>Default value</th><th>Description</th></tr>
<tr><td><b>Input setting</b></td><td><b>404</b></td><td><b>Input time constant setting (X0 to X3)</b></td><td>H0001</td><td>In the FP–M C16T: Enter the set value to change the input constant time. The input constant time corresponding to the set value is set to X0 to X3.
• Set value of input time constant
<table>
<tr><th>Input time constant</th><th>Set value of digit</th></tr>
<tr><td><b>1ms</b></td><td>H0</td></tr>
<tr><td><b>2ms</b></td><td>H1</td></tr>
<tr><td><b>4ms</b></td><td>H2</td></tr>
<tr><td><b>8ms</b></td><td>H3</td></tr>
<tr><td><b>16ms</b></td><td>H4</td></tr>
<tr><td><b>32ms</b></td><td>H5</td></tr>
<tr><td><b>64ms</b></td><td>H6</td></tr>
<tr><td><b>128ms</b></td><td>H7</td></tr>
</table>

• Digit and corresponding input (4 points)
404 = H000□
X0 to X3</td></tr>
</table>

<table>
<tr><th>Item</th><th>Address</th><th>Name</th><th>Default value</th><th>Description</th></tr>
<tr><td rowspan="5"><b>Input con-stant</b></td><td><b>404</b></td><td><b>Input time constant setting (X0 to X1F)</b></td><td>H1111</td><td rowspan="4">Sets the input constant time in 8–input units.
• Set value of input time constant
<table>
<tr><th>Input time constant</th><th>Set value of digit</th></tr>
<tr><td><b>1ms</b></td><td>H0</td></tr>
<tr><td><b>2ms</b></td><td>H1</td></tr>
<tr><td><b>4ms</b></td><td>H2</td></tr>
<tr><td><b>8ms</b></td><td>H3</td></tr>
<tr><td><b>16ms</b></td><td>H4</td></tr>
<tr><td><b>32ms</b></td><td>H5</td></tr>
<tr><td><b>64ms</b></td><td>H6</td></tr>
<tr><td><b>128ms</b></td><td>H7</td></tr>
</table>
• Set system registers 404, 405, 406, and 407, referring to the following:


Example: If you specify the input time constant for X0 to X7 as 4ms, input H1112 to system register 404.
System register 404
<table>
<tr><td><b>Bit position</b></td><td>15 · · 12</td><td>11 · · 8</td><td>7 · · 4</td><td>3 · · 0</td></tr>
<tr><td><b>Data input</b></td><td>0 0 0 1</td><td>0 0 0 1</td><td>0 0 0 1</td><td>0 0 1 0</td></tr>
<tr><td></td><td>H1
X18 to X1F</td><td>H1
X10 to X17</td><td>H1
X8 to XF</td><td>H2
X0 to X7</td></tr>
</table>
</td></tr>
<tr><td><b>405</b></td><td><b>Input time constant setting (X20 to X3F)</b></td><td>H1111</td></tr>
<tr><td><b>406</b></td><td><b>Input time constant setting (X40 to X5F)</b></td><td>H1111</td></tr>
<tr><td><b>407</b></td><td><b>Input time constant setting (X60 to X6F)</b></td><td>H0011</td></tr>
<tr><td><b>408, 409</b></td><td><b>Not used</b></td><td>——</td><td>——</td></tr>
</table>

Item	Address	Name	Default value	Description
Tool port setting	**410**	**Unit number setting for tool port (when connecting C–NET)**	K1	K1 to K32 (Unit No. 1 to 32)
	411	**Communication format settings for tool port** **Default setting items** **• Data length: 8 bits** **• Modem: Disabled** **The modem communication settings are available only for CPU version 2.7 or later and it setting are not available for FP–M C16 and FP1 C14/C16.**	H0	 Modem communication 0: Disabled 1: Enabled Data length (character bits) 0: 8 bits 1: 7 bits When connecting a modem, set the unit number to 1 with system register 410.
RS232C port setting	**412**	**Communication method setting for RS232C port**	K0	K0: RS232C port is not used. K1: Computer link communication (when connecting C–NET) K2: Serial data communication (general port)

<table>
<tr><th>Item</th><th>Address</th><th>Name</th><th>Default value</th><th>Description</th></tr>
<tr>
<td><b>RS232C port setting</b></td>
<td><b>413</b></td>
<td><b>Communication format settings for RS232C port</b>
<b>Default setting items</b>
<b>• Data length: 8 bits</b>
<b>• Parity check: With odd</b>
<b>• Stop bit: 1 bit</b>
<b>• Terminal code: C_R</b>
<b>• Start code: Without STX</b>
<b>(The settings for the header and the terminator in system register 413 become effective when system register 412 is set to K2 (GENERAL). If you select K1 (COMPTR LNK) or K0 (UNUSED), the settings are discarded.)</b></td>
<td>H3</td>
<td>

Start code (Bit position 6)
0: without STX
1: with STX
Terminal code (Bit position 5 & 4)
00: C_R
01: C_R + LF
11: EXT
Stop bit (Bit position 3)
0: 1 bit
1: 2 bits
Parity check (Bit position 2 & 1)
00: none
01: with odd
11: with even
Data length (Bit position 0)
0: 7 bits
1: 8 bits

Example: If you want to set the RS232C port as follows, input H13 to system register 413.
– Start code: without STX
– Terminal coder: C_R+LF
– Stop bit: 1 bit
– Parity check: with odd
– Data length: 8 bits

System register 413
<table>
<tr><th>Bit position</th><td>15 · · 12</td><td>11 · · 8</td><td>7 · · 4</td><td>3 · · 0</td></tr>
<tr><th>Data input</th><td>0 0 0 0</td><td>0 0 0 0</td><td>0 0 0 1</td><td>0 0 1 1</td></tr>
<tr><td></td><td></td><td></td><td>H1</td><td>H3</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><th>Item</th><th>Address</th><th>Name</th><th>Default value</th><th>Description</th></tr>
<tr><td rowspan="3"><b>RS232C port setting</b></td><td><b>414</b></td><td><b>Baud rate settings (for computer link and serial data communication)</b></td><td>K1</td><td><table><tr><th>Set value</th><th>Baud rate</th></tr><tr><td><b>K0</b></td><td>19,200bps</td></tr><tr><td><b>K1</b></td><td>9,600bps</td></tr><tr><td><b>K2</b></td><td>4,800bps</td></tr><tr><td><b>K3</b></td><td>2,400bps</td></tr><tr><td><b>K4</b></td><td>1,200bps</td></tr><tr><td><b>K5</b></td><td>600bps</td></tr><tr><td><b>K6</b></td><td>300bps</td></tr></table></td></tr>
<tr><td><b>415</b></td><td><b>Unit number setting for RS232C port (when connecting C–NET)</b></td><td>K1</td><td>K1 to K32 (Unit number 1to 32)</td></tr>
<tr><td><b>416</b></td><td><b>Modem compatibility setting for RS232C port</b></td><td>H0</td><td>• Settings:
H0: modem communication disabled
H8000: modem communication enabled
When modem communication is enabled, set system registers 412, 413, 415.
412: K1 Computer link communication
413: Set the communication format in order to set total number of bits in 10 bits.
(Example) Data length: 8 bits
Parity check: none
Stop bit: 1
415: K1 Unit No.1</td></tr>
<tr><td rowspan="2"><b>General port setting</b></td><td><b>417</b></td><td><b>Starting address setting for received buffer of serial data communication mode (Data register number)</b></td><td>K0</td><td>• Setting range:
C version FP–M 2.7k type and FP1 C24C/C40C types: K0 to K1660
C version FP–M 5k type and FP1 C56C/C72C types: K0 to K6144
For details about its usage, refer to <b>F144 (TRNS)</b> instruction.</td></tr>
<tr><td><b>418</b></td><td><b>Capacity setting for received buffer of serial data communication mode (Number of word)</b></td><td>K1660</td><td>• Setting range:
C version FP–M 2.7k type and FP1 C24C/C40C types: K0 to K1660 words
C version FP–M 5k type and FP1 C56C/C72C types: K0 to K6144 words
For details about its usage, refer to <b>F144 (TRNS)</b> instruction.</td></tr>
</table>

Appendix F

Glossary

Action Assignment

An action combines one sequence (created with the SFC–editor) with parts of the logic which are executed when a specific step is active. An action contains parts of the over–all logic. An action can be assigned to multiple steps and can be coded in FBD, LD or IL.

Body

A POU consists of a header and a body. The body contains the PLC program.

Data Type

Each variable is assigned a data type that determines its bit length. There are elementary (e.g. BOOL, WORD) and user–defined (e.g. ARRAY) data types.

Data Unit Type

A Data Unit Type (DUT) is a group of variables composed of several elementary data types. Such groups are used when data tables are edited.

Declaration

is the definition of Variables for global or local use.

EN (Enable) Input/ENO (Enable Out) Output

Many function blocks have an input and output variable of the data type BOOL in addition to the other input and output variables. The status of the ENO output always reflects the current status of the EN input.

F Instructions

are common Matsushita instructions. The P instructions function exactly the same way as the F instructions with the exception that they are only executed when a leading edge is detected.

Function

Functions are used within the definition of the user logic whenever a routine is needed, which, when executed, yields exactly one result. Since Functions do not access any internal memory, every invocation of one Function with identical input parameters always results in an identical value, the Function result.
As soon as a Function has been declared it can be accessed from any other Program Organization Unit of the User Logic.

Function Block

Function Blocks define both the algorithm as well as the data declaration of a part of the User Logic. Due to this definition the logic can be considered a class. Not the Function Block itself is invoked but several instances of this Function Block can be created, which can then be used separately. Each instance possesses its own copy of the data declaration memory, which provides the necessary data information for executing the Function Block functionality.

The private data declaration memory of a Function Block Instance persists from one invocation of this instance to the next one. This internal memory allows the implementation of incremental functionality by using Function Blocks.

As a consequence several invocations of one Function Block Instance with the same input variables will not necessarily yield the same results.

In comparison with Functions, Function Blocks allow you to define not only one but a set of output variables representing the Function Block results.

Substances of Function Blocks can be declared locally, for use within one POU. Declaring the instance of a Function Block within a POU defines the scope of this instance at the same time.

Function Block Diagram FBD

is a graphical language for programming connective logic. The individual Program Organization Unit’s Variables are connected with the inputs and outputs of function boxes. The connection represents a data flow between variables and inputs/outputs of function boxes.

A Function Block Diagram program is internally structured via Networks.

A Function Block Diagram network is defined by a connected graph of function boxes.

Function Block Instance

An object of the Function Block class

possesses its own copy of the Function Block's data declaration memory. This private data area is linked to the Function Block algorithm for this particular instance.

Global Variables
Global variables have physical addresses. They apply to the entire project and can be copied into the POU headers as VAR_EXTERNAL. The Global Variable List is found in the Project Navigator.

Header
A Program Organization Unit (POU) consists of a header and a body. In the header all variables used in the POU are listed and defined.

Identifier
is the symbolic name of a variable.

Input Variable
Input variables provide a function block/function with values with which calculations are carried out.

Instruction List IL
is a low level textual language which provides the capabilities for effective PLC programming. It is based on individual instructions which define one operation per instruction. Besides the Variables listed explicitly as arguments for an operation the actual value of the accumulator is used as an additional implicit argument. The result of an operation is also stored here after the execution of the appropriate instruction, thus providing a link between a preceeding instruction and one afterwards.

An Instruction List program is internally structured as an assembly of Networks.

Ladder Diagram LD
is a graphical language for programming connective logic. Similar to the Function Block Diagram capabilities, the individual Program Organization Unit's Variables are connected with the inputs and outputs of function boxes. In addition, Boolean connections can be drawn by using coils and contacts. This connection represents a Boolean signal flow.

A Ladder Diagram program is internally structued via Networks.

A Ladder Diagram network is defined by a connected graph of functions boxes linked with the lefthand power rail.

Local Variables
Local variables only apply to the POU in whose header they have been declared.

Logic
The complete PLC program defined by the user for solving the automation problem. The user logic is structured via Program Organization Units.

Network
A network belongs to a POU body and contains the logic (program).

Output Variable
Functions and function blocks write their results in output variables.

P Instructions
F instructions.

POU Pool
The POU Pool is located in the Project Navigator and contains all POUs that are part of the project.

Program
is similar to a Function Block with one implicit Function Block Instance. The differences between Programs and Function Blocks are:

Programs are only allowed on top of a Program Organization Unit invocation hierarchy (i.e. a program may not be invoked from another Program Organization Unit)

Directly represented Variables can be used for defining a Program

Program Organization Unit (POU)
Program Organization Units are used for structuring the complete user logic. Individual Units may invoke other ones, however a recursive POU structure is not allowed.

Program Organization Units are either defined as standard by default or user specific due to the specific automation problem to be solved by the User Logic.

FPWIN Pro differentiates between the Program Organization Unit Header (which contains the Declaration part of the Program

Organization Unit) and the Program Organization Unit Body (which contains the Program Organization Unit's algorithm).

Due to different requirements for the solution of a sub–problem, different typs of POUs are provided.

The different Program Organization Unit types are Functions, Function Blocks and Programs.

Project
The project represents the top level of the hierarchy in Control FPWIN Pro. It contains the entire task for the controller.

Sequential Function Chart SFC
consists of the basic elements steps and transitions. While steps represent a specific state during the execution of a POU, a transition allows the definition of the conditions for changing from one state to the next state.

Using either parallel or alternative branches you can complement several types of SFC sequences.

Specific connective logic program code can be associated to the steps via actions by using the appropriate languages Function Block Diagram, Ladder Diagram, Structured Text and Instruction List.

Structured Text
is a text–based editor exempt from normal syntax. ST is a high–level language that allows you to write complex programs and control structures. It is available for all PLCs and requires no more resources, e.g. steps, labels or calls, than other editors while doing comparable programming.

Task
defines the moment (and other execution parameters) of program execution. A POU of type program contains the logic, i.e., it defines what has to be done. The association of a program to a task defines the moment of the logic's execution.

Variable
enables the association of a specifier to a specific memory area. Due to different requirements, data can be of different types. Variables can be either global, for use within the entire user program, or local, being restricted to the POU in which it has been defined.

Index

Record of Changes

Manual No.	Date	Description of Changes
ACGM0130END V1.0	June 1998	First edition
ACGM0130END V1.1	Oct. 1999	Updated, appendix, glossary, new commands: **IEC Functions**: INT_TO_REAL, DINT_TO_TIME, DINT_TO_REAL, DWORD_TO_TIME, REAL_TO_INT, REAL_TO_DINT, TIME_TO_DINT, TIME_TO_DWORD, TRUNC_TO_INT, TRUNC_TO_DINT, SQRT, SIN, ASIN, COS, ACOS, TAN, ATAN, LN, LOG, EXP, EXPT, MUL_TIME_DINT, MUL_TIME_REAL, DIV_TIME_DINT, DIV_TIME_REAL; **Matsushita Instructions**: CT, DF, DFN, ICTL, JP, KEEP, LBL, LOOP, LSR, MC, MCE, TM_1ms,TM_10ms, TM_100ms, TM_1s, F12_EPRD, EEPROM read from memory P13_EPWT, EEPROM write to memory F327_INT, Floating point data 16–bit integer data (the largest integer not exceeding the floating point data) F328_DINT, Floating point data 32–bit integer data (the largest integer not exceeding the floating point data) F333_FINT, Rounding the first decimal point down F334_FRINT, Rounding the first decimal point off F335_FSIGN, Floating point data sign changes (negative/positive conversion) F337_RAD, Conversion of angle units (Degrees Radians) F338_DEG, Conversion of angle units (Radians Degrees) F355_PID, PID processing instruction.
ACGM0130END V2.0	Feb. 2001	Revision of several commands including F144, F168, F169, F170, F70–F83, CT Inclusion of F0_MV as used to initialize DT9052, SET/RESET. Name change of NAiS Control to FPWIN Pro. Additional appendices: data registers, relays, memory areas and system registers. Layout changes
ACGM0130V3.0END	Oct. 2001	Update for release of FPWIN Pro Version 4.0 Error removal Addition of ST examples
ACGM0130V3.1END	Nov. 2001	Selected IEC commands with STRING functionality added
ACGM0130V3.2END	May 2002	Linear page numbering, instruction indexing in header, minor error corrections (e.g. F355_PID, number of ARRAY elements)

GLOBAL NETWORK

North America	Europe	Asia Pacific	China	Japan
Aromat Corporation	**Matsushita Electric Works Group**	**Matsushita Electric Works (Asia Pacific)**	**Matsushita Electric Works**	**Matsushita Electric Works Ltd. Automation Controls Group**

Europe

■ **Austria** — **Matsushita Electric Works Austria GmbH**
Stojanstraße 12, 2344 Maria Enzersdorf, Austria, Tel. (02236) 2 68 46, Fax (02236) 46133, http://www.matsushita.at

■ **Benelux** — **Matsushita Electric Works Benelux B. V.**
De Rijn 4, (Postbus 211), 5684 PJ Best, (5680 AE Best), Netherlands, Tel. (0499) 37 2727, Fax (0499) 372185, http://www.matsushita.nl

■ **France** — **Matsushita Electric Works France S.A.R.L.**
B.P. 44, 91371 Verrières le Buisson CEDEX, France, Tel. 01 60 13 57 57, Fax 01 60 13 57 58, http://www.matsushita–france.fr

■ **Germany** — **Matsushita Electric Works Deutschland GmbH**
Rudolf–Diesel–Ring 2, 83607 Holzkirchen, Germany, Tel. (08024) 648–0, Fax (08024) 648–555, http://www.matsushita.de

■ **Ireland** — **Matsushita Electric Works Ltd., Irish Branch Office**
Waverley, Old Naas Road, Bluebell, Dublin 12, Republic of Ireland, Tel. (01) 460 09 69, Fax (01) 460 11 31

■ **Italy** — **Matsushita Electric Works Italia s.r.l.**
Via del Commercio 3–5 (Z.I. Ferlina), 37012 Bussolengo (VR), Italy, Tel. (045) 675 27 11, Fax (045) 670 04 44, http://www.matsushita.it

■ **Portugal** — **Matsushita Electric Works Portugal, Portuguese Branch Office**
Avda 25 de Abril, Edificio Alvorada 5º E, 2750 Cascais, Portugal, Tel. (351) 1482 82 66, Fax (351) 1482 74 21

■ **Scandinavia** — **Matsushita Electric Works Scandinavia AB**
Sjöängsvägen 10, 19272 Sollentuna, Sweden, Tel. +46 8 59 47 66 80, Fax (+46) 8 59 47 66 90, http://www.mac–europe.com

■ **Spain** — **Matsushita Electric Works España S.A.**
Parque Empresarial Barajas, San Severo, 20, 28042 Madrid, Spain, Tel. (91) 329 38 75, Fax (91) 329 29 76

■ **Switzerland** — **Matsushita Electric Works Schweiz AG**
Grundstrasse 8, 6343 Rotkreuz, Switzerland, Tel. (041) 799 70 50, Fax (041) 799 70 55, http://www.matsushita.ch

■ **United Kingdom** — **Matsushita Electric Works UK Ltd.**
Sunrise Parkway, Linford Wood East, Milton Keynes, MK14 6LF, England, Tel. (01908) 231 555, Fax (01908) 231 599, http://www.matsushita.co.uk

North & South America

■ **USA** — **Aromat Corporation Head Office USA**
629 Central Avenue, New Providence, N.J. 07974, USA, Tel. 1–908–464–3550, Fax 1–908–464–8513, http://www.aromat.com

Asia

■ **China** — **Matsushita Electric Works, Ltd. China Office**
2013, Beijing Fortune, Building 5, Dong San Huan Bei Lu, Chaoyang District, Beijing, China, Tel. 86–10–6590–8646, Fax 86–10–6590–8647

■ **Hong Kong** — **Matsushita Electric Works Ltd. Hong Kong**
Rm1601, 16/F, Tower 2, The Gateway, 25 Canton Road, Tsimshatsui, Kowloon, Hong Kong, Tel. (852) 2956–3118, Fax (852) 2956–0398

■ **Japan** — **Matsushita Electric Works Ltd. Automation Controls Group**
1048 Kadoma, Kadoma–shi, Osaka 571–8686, Japan, Tel. 06–6908–1050, Fax 06–6908–5781, http://www.mew.co.jp/e–acg/

■ **Singapore** — **Matsushita Electric Works Pte. Ltd. (Asia Pacific)**
101 Thomson Road, #25–03/05, United Square, Singapore 307591,Tel. (65) 255–5473, Fax (65) 253–5689

Specifications are subject to change without notice.

Printed in Europe